# Learning the semantics of structured data sources

Mohsen Taheriyan *, Craig A. Knoblock, Pedro Szekely, José Luis Ambite

*University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA*

A B S T R A C T

Information sources such as relational databases, spreadsheets, XML, JSON, and Web APIs contain a tremendous amount of structured data that can be leveraged to build and augment knowledge graphs. However, they rarely provide a semantic model to describe their contents. Semantic models of data sources represent the implicit meaning of the data by specifying the concepts and the relationships within the data. Such models are the key ingredients to automatically publish the data into knowledge graphs. Manually modeling the semantics of data sources requires significant effort and expertise, and although desirable, building these models automatically is a challenging problem. Most of the related work focuses on semantic annotation of the data fields (source attributes). However, constructing a semantic model that explicitly describes the relationships between the attributes in addition to their semantic types is critical.

We present a novel approach that exploits the knowledge from a domain ontology and the semantic models of previously modeled sources to automatically learn a rich semantic model for a new source. This model represents the semantics of the new source in terms of the concepts and relationships defined by the domain ontology. Given some sample data from the new source, we leverage the knowledge in the domain ontology and the known semantic models to construct a weighted graph that represents the space of plausible semantic models for the new source. Then, we compute the top $k$ candidate semantic models and suggest to the user a ranked list of the semantic models for the new source. The approach takes into account user corrections to learn more accurate semantic models on future data sources. Our evaluation shows that our method generates expressive semantic models for data sources and services with minimal user input. These precise models make it possible to automatically integrate the data across sources and provide rich support for source discovery and service composition. They also make it possible to automatically publish semantic data into knowledge graphs.

© 2015 Elsevier B.V. All rights reserved.

## 1. Introduction

Knowledge graphs have recently emerged as a rich and flexible representation of domain knowledge. Nodes in this graph represent the entities and edges show the relationships between the entities. Large companies such as Google and Microsoft employ knowledge graphs as a complement for their traditional search methods to enhance the search results with semantic-search information. Linked Open Data (LOD) is an ongoing effort in the Semantic Web community to build a massive public knowledge graph. The goal is to extend the Web by publishing various open datasets as RDF on the Web and then linking data items to other useful information from different data sources. With linked data, starting from a certain point in the graph, a person or machine can explore the graph to find other related data. The focus of this work is the first step of publishing linked data, automatically publishing datasets as RDF using a common domain ontology.

A large amount of data in LOD comes from structured sources such as relational databases and spreadsheets. Publishing these sources into LOD involves constructing *source descriptions* that represent the intended meaning of the data by specifying mappings between the sources and the *domain ontology* [1]. A domain ontology is a formal model that represents the concepts within a domain and the properties and interrelationships of those concepts. In this context, what is meant by a source description is a schema mapping from the source to an ontology. We can represent this mapping as a *semantic network* with ontology classes as the nodes and ontology properties as the links between the nodes. This network, also called a *semantic model*, describes the source in terms of the concepts and relationships defined by the domain ontology. Fig. 1 depicts a semantic model for a sample data source including

* Corresponding author.
*E-mail addresses:* mohsen@isi.edu (M. Taheriyan), knoblock@isi.edu (C.A. Knoblock), pszekely@isi.edu (P. Szekely), ambite@isi.edu (J.L. Ambite).
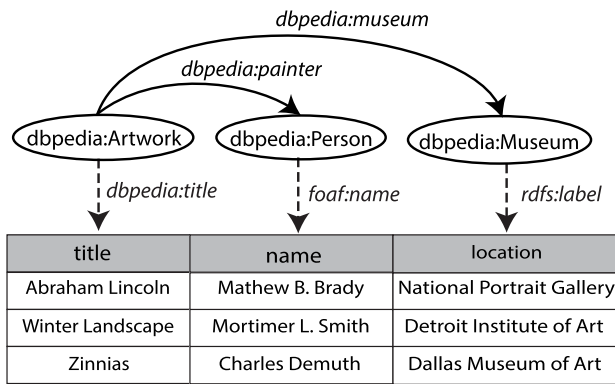
**Fig. 1.** The semantic model of a sample data source containing information about paintings.

information about some paintings. This model explicitly represents the meaning of the data by mapping the source to the DBpedia[1] and FOAF[2] ontologies. Knowing this semantic model enables us to publish the data in the table into the LOD knowledge graph.

One step in building a semantic model for a data source is *semantic labeling*, determining the *semantic types* of its data fields, or *source attributes*. That is, each source attribute is labeled with a class and/or a data property of the domain ontology. In our example in Fig. 1, the semantic types of the first, second, and third columns are *title* of *Artwork*, *name* of *Person*, and *label* of *Museum* respectively. However, simply annotating the attributes is not sufficient. Unless the relationships between the columns are explicitly specified, we will not have a precise model of the data. In our example, a *Person* could be the *owner*, *painter*, or *sculptor* of an *Artwork*, but in the context of the given source, only *painter* correctly interprets the relationship between *Artwork* and *Person*. In the correct semantic model, *Museum* is connected to *Artwork* through the link *museum*. Other models may connect *Museum* to *Person* instead of *Artwork*. For instance, *Person* could be *president*, *owner*, or *founder* of *Museum*, or *Museum* could be *employer* or *workplace* of *Person*. To build a semantic model that fully recovers the semantics of the data, we need a second step that determines the relationships between the source attributes in terms of the properties in the ontology.

Manually constructing semantic models requires significant effort and expertise. Although desirable, generating these models automatically is a challenging problem. In Semantic Web research, there is much work on mapping data sources to ontologies [2–13], but most focus on semantic labeling or are very limited in automatically inferring the relationships. Our goal is to construct semantic models that not only include the semantic types of the source attributes, but also describe the relationships between them.

In this paper, we present a novel approach that exploits the knowledge from a domain ontology and *known semantic models* of sources in the same domain to automatically learn a rich semantic model for a new source. The work is inspired by the idea that different sources in the same domain often provide similar or overlapping data and have similar semantic models. Given sample data from the new source, we use a labeling technique [14] to annotate each source attribute with a set of candidate semantic types from the ontology. Next, we build a weighted directed graph from the known semantic models, learned semantic types, and the domain ontology. This graph models the space of plausible

semantic models. Then, we find the most promising mappings from the source attributes to the nodes of the graph, and for each mapping, we generate a candidate model by computing the minimal tree that connects the mapped nodes. Finally, we score the candidate models to prefer the ones formed with more coherent and frequent patterns.

This work builds on top of our previous work on learning semantic models of sources [15,16]. The central data structure of our approach to learn a semantic model for a new source is a graph built on top of the known semantic models. In the previous work, we add a new component to the graph for each known semantic model. If two semantic models are very similar to each other and they only differ in one link, for example, we will still have two different components for them in the graph. The graph grows as the number of known semantic models grows, which makes computing the semantic models inefficient if we have a large set of known semantic models. In this paper, we extend our previous work to make it scale to a large number of semantic models. We present a new algorithm that constructs a much more compact graph by merging overlapping segments of the known semantic models. The new technique significantly reduces the size of the graph in terms of the number of nodes and links. Consequently, it considerably decreases the number of possible mappings from the source attributes to the nodes of the graph. It also makes computing the minimal tree that connects the nodes of the candidate mappings in the graph more efficient. The new method to build the graph changes our algorithms to compute and rank the candidate semantic models.

The main contribution of this paper is a scalable approach that exploits the structure of the domain ontology and the known semantic models to build semantic models of new sources. We evaluated our approach on a set of museum data sources modeled using two well-known *data models* in the cultural heritage domain: Europeana Data Model (EDM) [17], and CIDOC Conceptual Reference Model (CIDOC-CRM) [18]. A data model standardizes how to map the data elements in a domain to a set of domain ontologies. The evaluation shows that our approach automatically generates high-quality semantic models that would have required significant user effort to create manually. It also shows that the semantic models learned using both the domain ontology and the known models are approximately 70% more accurate than the models learned with the domain ontology as the only background knowledge. The generated semantic models are the key ingredients to automate tasks such as source discovery, information integration, and service composition. They can also be formalized using mapping languages such as R2RML[19], which can be used for converting data sources into RDF and publishing them into the Linked Open Data (LOD) cloud or any other knowledge graph.

We have implemented our approach in Karma [20], our data modeling and integration framework.[3] Users can import data from a variety of sources including relational databases, spreadsheet, XML files and JSON files into Karma. They can also import the domain ontologies they want to use for modeling the data. The system then automatically suggests a semantic model for the loaded source. Karma provides an easy to use graphical user interface to let users interactively refine the learned semantic models if needed. Once a semantic model is created for the new source, users can publish the data as RDF by clicking a single button. Szekely et al. [21] used Karma to model the data from Smithsonian American Art Museum[4] and then publish it into the

| title | creationDate | name | type |
|---|---|---|---|
| Stream in the Mountains | 1825 | George Inness | Paintings |
| Lady Godiva | 1856 | Anne Whitney | Sculpture |
| Prodigal Son | 1934 | Thomas Hart Benton | Paintings |

(a) dma(title, creationDate, name, type).

| name | artist | year | image |
|---|---|---|---|
| Neil Armstrong | Louis Glanzman | 1969 | http://npgportraits.si.edu/ |
| David Baltimore | Jon R. Friedman | 2006 | http://npgportraits.si.edu/ |
| Henry Larcom Abbo | Nahum Ball Onth | 1857 | http://npgportraits.si.edu/ |

(b) npg(name, artist, year, image).

| title | credit | classification | name | imageURL |
|---|---|---|---|---|
| Indian Telegrap | Formerly in the | Paintings | Mortimer L. Smith | http://www.dia.org/us |
| The Seashore | Gift of the artist, | Paintings | Leon Dabo | http://www.dia.org/us |
| Land and Sea | Mrs. John L. Ga | Paintings | William Mark Fishe | http://www.dia.org/us |

(c) dia(title, credit, classification, name, imageURL).

**Fig. 2.** Sample data from three museum sources: (a) Dallas Museum of Art, (b) National Portrait Gallery, and (c) Detroit Institute of Art.

Linked Open Data cloud. Karma is also able to build semantic models for Web services and then exploits the created semantic models to build APIs that directly communicate at the semantic level [22–24].

## 2. Motivating example

We explain the problem of learning semantic models by giving a concrete example that will be used throughout this paper to illustrate different steps of our approach. In this example, the goal is to model a set of museum data sources using EDM,[5] AAC,[6] SKOS,[7] Dublin Core Metadata Terms,[8] FRBR,[9] FOAF, ORE,[10] and ElementsGr2[11] ontologies and then use the created semantic models to publish their data as RDF [21]. Suppose that we have three data sources. The first source is a table containing information about artworks in the Dallas Museum of Art[12] (Fig. 2(a)). We formally write the signature of this source as *dma(title, creationDate, name, type)* where *dma* is the name of the source and *title*, *creationDate*, *name*, and *type* are the names of the source attributes (columns). The second source, *npg*, is a CSV file including the data of some of the portraits in the National Portrait Gallery[13] (Fig. 2(b)), and the third data source, *dia*, has the data of the artworks in the Detroit Institute of Art[14] (Fig. 2(c)).

Fig. 3 shows the correct semantic model of the sources *dma*, *npg*, and *dia* created by experts in the museum domain. A semantic model of the source *s*, called *sm(s)*, is a directed graph containing two types of nodes. *Class nodes* (ovals) correspond to classes in the ontology, and *data nodes* (rectangles) correspond to the source attributes (labeled with the attribute names). The links in the graph are associated with ontology properties. The particular link *karma:uri* from a class node, which represents an ontology class, to a data node, which represents a source attribute, denotes that the attribute values are the URIs of the class instances. For instance, in Fig. 3(b), the values of the column *image* in the source *npg* are the URIs of the instances of the class *edm:WebResource*.

As discussed earlier, automatically building the semantic models is difficult. Machine learning methods can help us in assigning semantic types to the attributes by looking into the attributes values, however, these methods are error prone when similar data

values have different semantic types. For example, from just the data values of the attribute *creationDate* in the source *dma*, it is hard to say whether it is the creation date of *aac:CulturalHeritageObject* or it is the birthdate of a *aac:Person*. Extracting the relationships between the attributes is a more complicated problem. There might be multiple paths connecting two classes in the ontology and we do not know which one captures the intended meaning of the data. For instance, there are several paths in the domain ontology connecting *aac:CulturalHeritageObject* to *aac:Person*, but in the context of the source *dma*, only the link *dcterms:creator* represents the correct meaning of the source. As another example, the attributes *artist* and *name* in the source *npg* are both labeled with name of *Person*, nevertheless, how can we decide whether these two attributes are different names of one person or they belong to two distinct individuals? In general, the ontology defines a large space of possible semantic models and without additional context, we do not know which one describes the source more precisely.

Now, assume that the correct semantic models of the sources *dma* and *npg* are given. Can we leverage these known semantic models to build a semantic model for a new source such as *dia*? In the next section, we present a scalable and automated approach that exploits the known semantic models *sm(dma)* and *sm(npg)* to limit the search space and learn a semantic model *sm(dia)* for the new source *dia*.

## 3. Learning semantic models

We now formally state the problem of learning semantic models of data sources. Let $O$ be the domain ontology[15] and $\{sm(s_1), sm(s_2), \ldots, sm(s_n)\}$ is a set of known semantic models corresponding to the data sources $\{s_1, s_2, \ldots, s_n\}$. Given sample data from a new source $s(a_1, a_2, \ldots, a_m)$ called the *target source*, in which $\{a_1, a_2, \ldots, a_m\}$ are the source attributes, our goal is to automatically compute a semantic model $sm(s)$ that captures the intended meaning of the source $s$. In our example, *sm(dma)* and *sm(npg)* are the known semantic models, and the source *dia* is the new source for which we want to automatically learn a semantic model.

The main idea is that data sources in the same domain usually provide overlapping data. Therefore, we can leverage attribute relationships in known semantic models to hypothesize attribute relationships for new sources. One of the metrics helping us to infer relationships between the attributes of a new source is the popularity of the links between the semantic types in the set of known models. Nevertheless, simply using link popularity to connect a set of nodes would lead to myopic decisions that select links that appear frequently in other models without taking into account how these nodes are connected to other nodes in the given

(a) *sm(dma)*: semantic model of the source *dma*.

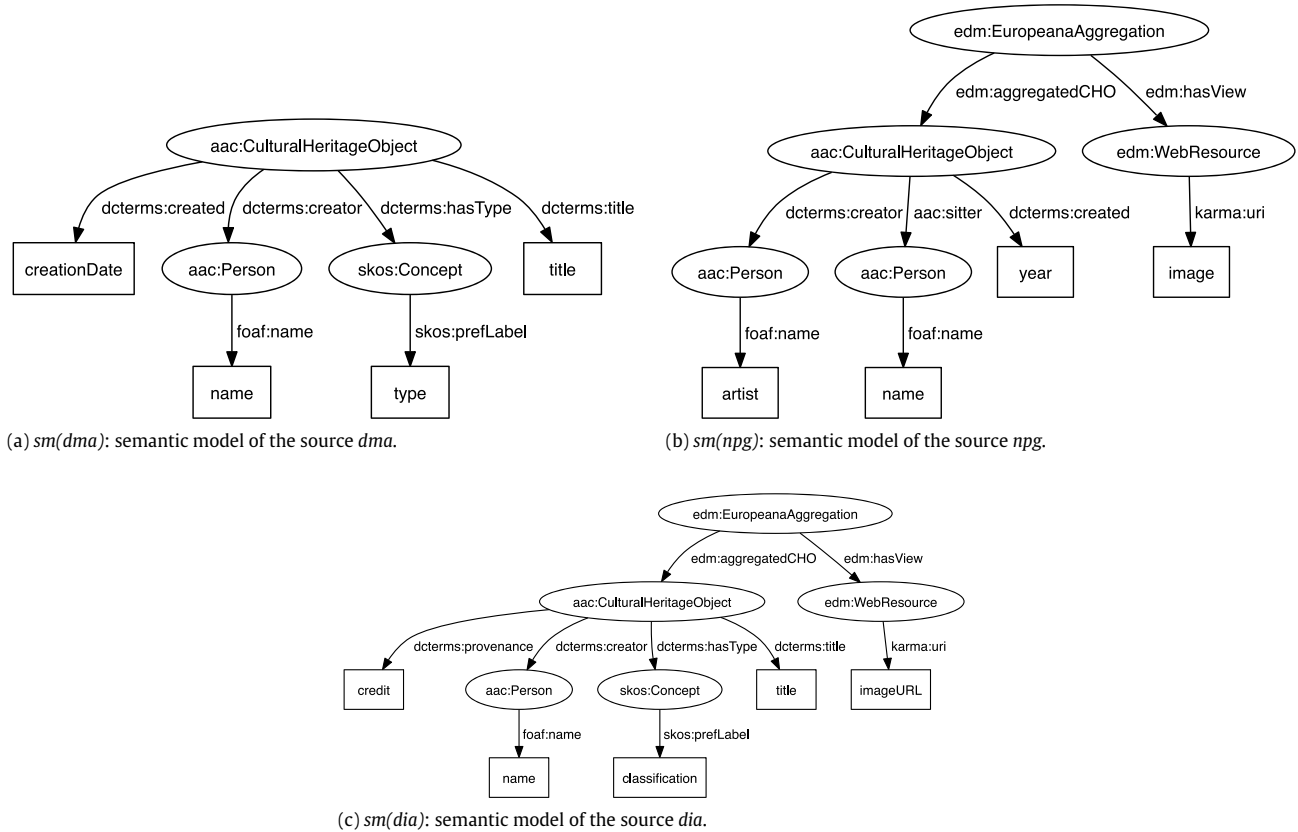(b) *sm(npg)*: semantic model of the source *npg*.

(c) *sm(dia)*: semantic model of the source *dia*.

**Fig. 3.** Semantic models of the example data sources created by experts in the museum domain. Class nodes (ovals) and links correspond to classes and properties in the ontology (prefixed by the ontology namespace). The particular link *karma:uri* in (b) simply means that the values of the attribute *image* in the source *npg* are the URIs of the instances of the class *edm:WebResource*.

models. Suppose that we have a set of 5 known semantic models. One of these models contains the link *painter* between *Artwork* and *Person* and the link *museum* between *Artwork* and *Museum* (similar to the example in Fig. 1). The other 4 models do not contain the type *Artwork*, but they include the link *founder* from *Museum* to *Person*. If a given new source contains the types *Artwork*, *Museum*, and *Person*, just using the link popularity yields to an incorrect model. Our approach takes into account the coherence of the patterns in addition to their popularity, and this is more complicated to do.

Our approach to learn a semantic model for a new source has four steps: (1) Using sample data from the new source, learn the semantic types of the source attributes. (2) Construct a graph from the known semantic models, augmented with nodes and links corresponding to the learned semantic types and ontology paths connecting nodes of the graph. (3) Compute the candidate mappings from the source attributes to the nodes of the graph. (4) Finally, build candidate semantic models for the candidate mappings, and rank the generated models.

### 3.1. Learning semantic types of source attributes

The first step to model the semantics of a new source is to recognize the semantic types of its data. We call this step semantic labeling, which involves annotating the source columns with classes or properties in an ontology. The objective of this step is to assign semantic types to source attributes. We formally define a semantic type to be either an ontology class $\langle class\_uri \rangle$ or a pair consisting of a domain class and one of its data properties $\langle class\_uri, property\_uri \rangle$. We use a class as a semantic type for attributes whose values are URIs for instances of a class and for attributes containing automatically-generated database keys that can also be modeled as instances of a class. We use

a domain/data property pair as a semantic type for attributes containing literal values. For example, the semantic types of the attributes *imageURL* and *classification* in the source *dia* are respectively $\langle edm:WebResource \rangle$ and $\langle skos:Concept, skos:prefLabel \rangle$.

While syntactic information about data sources such as attribute names or attribute types (string, int, date, …) may give the system some hints to discover semantic types, they are often not sufficient, e.g., name of the first field in the source *dia* is *title* and we do not know whether this is a title of a book, song, or an artwork. Moreover, in many cases, attribute names are used in abbreviated forms, e.g., *dob* rather than *birthdate*.

We employ the technique proposed by Krishnamurthy et al. [14] to learn semantic types of source attributes. Their approach focuses on learning the semantic types from the data rather than the attribute names. It learns a semantic labeling function from a set of sources that have been manually labeled. When presented with a new source, the learned semantic labeling function can automatically assign semantic types to each attribute of the new source. The training data consists of a set of semantic types and each semantic type has a set of data values and attribute names associated with it. Given a new set of data values from a new source, the goal is to predict the top $k$ candidate semantic types along with confidence scores using the training data.

If the data values associated with a source attribute $a_i$ are textual data, the labeling algorithm uses the *cosine similarity* between TF/IDF vectors of the labeled documents and the input document to predict candidate semantic types. The set of data values associated with each textual semantic type in the training data is treated as a document, and the input document consists of the data values associated with $a_i$. For attributes with numeric data, the algorithm uses *statistical hypothesis testing* [25] to analyze the distribution of numeric values. The intuition is that the distribution

**Table 1**
Top two learned semantic types for the attributes of the source *dia*.

| Attribute | Candidate semantic types |
|---|---|
| *title* | $\langle aac{:}CulturalHeritageObject, dcterms{:}title \rangle^{0.49}$ |
| | $\langle aac{:}CulturalHeritageObject, rdfs{:}label \rangle^{0.28}$ |
| *credit* | $\langle aac{:}CulturalHeritageObject, dcterms{:}provenance \rangle^{0.83}$ |
| | $\langle aac{:}Person, ElementsGr2{:}note \rangle^{0.06}$ |
| *classification* | $\langle skos{:}Concept, skos{:}prefLabel \rangle^{0.58}$ |
| | $\langle skos{:}Concept, rdfs{:}label \rangle^{0.41}$ |
| *name* | $\langle aac{:}Person, foaf{:}name \rangle^{0.65}$ |
| | $\langle foaf{:}Person, foaf{:}name \rangle^{0.32}$ |
| *imageURL* | $\langle foaf{:}Document \rangle^{0.47}$ |
| | $\langle edm{:}WebResource \rangle^{0.40}$ |

of values in each semantic type is different. For example, the distribution of temperatures is likely to be different from the distribution of weights. The training data here consists of a set of numeric semantic types and each semantic type has a sample of numeric data values. At prediction time, given a new set of numeric data values (query sample), the algorithm performs statistical hypothesis tests between the query sample and each sample in the training data.

Once we apply this labeling method, it generates a set of candidate semantic types for each source attribute, each with a confidence value. Our algorithm then selects the top $k$ semantic types for each attribute as an input to the next step of the process. Thus, the output of the labeling step for $s(a_1, a_2, \ldots, a_m)$ is $T = \{(t_{11}^{p_{11}}, \ldots, t_{1k}^{p_{1k}}), \ldots, (t_{m1}^{p_{m1}}, \ldots, t_{mk}^{p_{mk}})\}$, where in $t_{ij}^{p_{ij}}$, $t_{ij}$ is the $j$th semantic type learned for the attribute $a_i$ and $p_{ij}$ is the associated confidence value which is a decimal value between 0 and 1. Table 1 lists the candidate semantic types for the source *dia* considering $k = 2$.

As we can see in Table 1, the semantic labeling method prefers $\langle foaf{:}Document \rangle$ for the semantic type of the attribute *imageURL*, while according to the correct model (Fig. 3(c)), $\langle edm{:}WebResource \rangle$ is the correct semantic type. We will show later how our approach recovers the correct semantic type by considering coherence of structure in computing the semantic models.

### 3.2. Building a graph from known semantic models, semantic types, and domain ontology

So far, we have tagged the attributes of *dia* with a set of candidate semantic types. To build a complete semantic model we still need to determine the relationships between the attributes. We leverage the knowledge of the known semantic models to discover the most popular and coherent patterns connecting the candidate semantic types.

The central component of our method is a directed weighted graph $G$ built on top of the known semantic models and expanded using the semantic types $T$ and the domain ontology $O$. Similar to a semantic model, $G$ contains both class nodes and data nodes and links. The links correspond to properties in $O$ and there are weights on the links. Algorithm 1 shows the steps to build the graph. Our algorithm has three parts: (1) adding the known semantic models, $sm(dma)$ and $sm(npg)$ (Algorithm 2); (2) adding the semantic types learned for the target source (Algorithm 3); and (3) expanding the graph using the domain ontology $O$ (Algorithm 4).

**Adding known semantic models**: Suppose that we want to add $sm(s_i)$ to the graph. If the graph is empty, we simply add all the nodes and links in $sm(s_i)$ to $G$, otherwise we merge the nodes and links of $sm(s_i)$ into $G$ by adding the nodes and links that do

---

**Algorithm 1** *Construct Graph* $G = (V, E)$

  **Input:**
      - Known Semantic Models $M = \{sm_1, \ldots, sm_n\}$,
      - Attributes(s) $A = \{a_1, \ldots, a_m\}$
      - Semantic Types $T = \{(t_{11}^{p_{11}}, \ldots, t_{1k}^{p_{1k}}), \ldots, (t_{m1}^{p_{m1}}, \ldots, t_{mk}^{p_{mk}})\}$
      - Ontology $O$
  **Output:** Graph $G = (V, E)$

1: ADDKNOWNMODELS($G, M$)
2: ADDSEMANTICTYPES($G, T$)
3: ADDONTOLOGYPATHS($G, O$)

  **return** $G$

---

not exist in $G$. When adding a new node or link, we tag it with a unique identifier (e.g., $s_i$, name of the source) indicating that the node/link exist in $sm(s_i)$. If a node or link already exists in the graph, we just add the identifier $s_i$ to its tags. The nodes and the links that are added in this step are shown with the black color in Fig. 4. In order to easily refer to the nodes of the figure in the text, we assign a unique name to each node. The name of a node is written with small font at the left side of the node. For example, the node with the label *edm:EuropeanaAggregartion* is named $n_1$. The orange and green tags below the labels of the black links are the identifiers indicating the semantic model(s) supporting the links. For instance, the link *dcterms:creator* from $n_2$ (*aac:CulturalHeritageObject*) to $n_7$ (*aac:Person*) is tagged with both *dma* and *npg*, because it exists in both $sm(dma)$ and $sm(npg)$. For readability, we have not put the tags of the nodes in Fig. 4.

Although merging a semantic model into $G$ looks straightforward, there are difficulties when the semantic model or the graph include multiple class nodes with the same label. Suppose that $G$ already includes two class nodes $v_1$ and $v_2$ both labeled with *Person* connected by the link *isFriendOf*. Now, we want to add a semantic model including the link *worksFor* from *Person* to *Organization*. Assuming $G$ does not have a class node with the label *Organization*, we add a new class node $v_3$ to $G$. Now, the question is where to put the link *worksFor*, between $v_1$ and $v_3$, or $v_2$ and $v_3$. One option is to duplicate the link by adding a link between each pair and then assign different tags to the added links. This approach slows down the process of building the graph, and because it can yield a graph with a large number of links, our algorithm to compute the candidate semantic models would be inefficient too. Therefore, we adopt a different strategy; if there is more than one node in the graph matching a node in the semantic model, we select the one having more tags. This heuristic creates a more compact graph and makes the whole algorithm faster, while not having much impact on the results.

Algorithm 2 illustrates the details of our method to add a semantic model $sm(s_i)$ to $G$:

1. [line 2]: Let $H$ be a HashMap keeping the mappings from the nodes in $sm(s_i)$ to the nodes in $G$. The key of each entry in $H$ is a node in $sm(i)$ and its value is a node in $G$.
2. [lines 3–13]: For each class node $v$ in $sm(s_i)$, we search the graph to see if $G$ includes a class node with the same label. If no such node exists in the graph, we simply add a new node to the graph. It is possible that $sm(s_i)$ contains multiple class nodes with the same label, for instance, a model including the link *isFriendOf* from one *Person* to another *Person*. In this case, we make sure that $G$ also has at least the same number of class nodes with that label. For example, if $G$ only has one *Person*, we add another class node with the label *Person*. Once we added the required class nodes to the graph, we map the class nodes in the model to the class nodes in the graph. If $G$ has multiple class nodes with the same label, we select the one that is tagged by larger number of known semantic models. We add an entry to $H$ with $v$ as the key and the mapped node ($v'$) as the value.
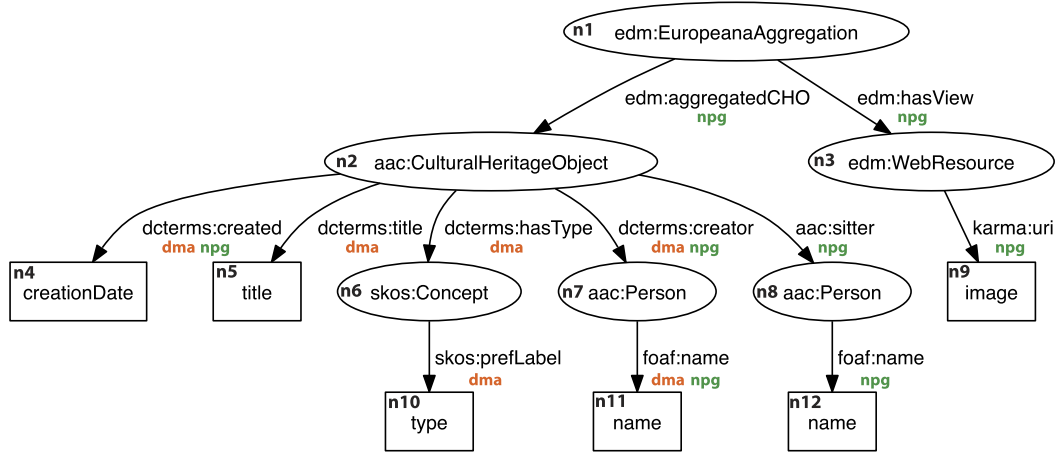
**Fig. 4.** The graph $G$ after adding the known semantic models $sm(dma)$ and $sm(npg)$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

---

**Algorithm 2** *Add Known Semantic Models to G*

1: **function** ADDKNOWNMODELS($G,M$)

2:     $H = HashMap\langle node, node \rangle$
    ▷ H keys: nodes in $sm_i$
    ▷ H values: matched nodes in $G$

3:     **for** each $sm_i \in M$ **do**
4:         **for** each class node $v$ in $sm_i$ **do**
5:             $l_v \leftarrow$ label of $v$
6:             $c_1 \leftarrow$ number of class nodes in $sm_i$ with label $l_v$
7:             $c_2 \leftarrow$ number of class nodes in $G$ with label $l_v$
8:             add $(c_1 - c_2)$ class nodes with label $l_v$ to $G$
9:             $matched\_nodes \leftarrow$ class nodes with label $l_v$ in $G$
10:            $unmapped\_nodes \leftarrow matched\_nodes - values(H)$
11:            $v' \leftarrow$ the node with largest tag set in $unmapped\_nodes$
12:            add $\langle v, v' \rangle$ to $H$
13:        **end for**

14:        **for** each link $e$ from a class node $u$ to a **data** node $v$ in $sm_i$ **do**
15:            $l_e \leftarrow$ label of $e$
16:            $u' \leftarrow H(u)$
17:            **if** $u'$ has an outgoing link with label $l_e$ **then**
18:                $v' \leftarrow$ target of the link with label $l_e$
19:            **else**
20:                add a new data node $v'$ to $G$
21:            **end if**
22:            add $\langle v, v' \rangle$ to $H$
23:        **end for**

24:        $w_l \leftarrow 1$
25:        **for** each link $e$ from $u$ to $v$ in $sm_i$ **do**
26:            $u' \leftarrow H(u)$
27:            $v' \leftarrow H(v)$
28:            **if** there is $e'$ from $u'$ to $v'$ with $l_{e'} = l_e$ in $G$ **then**
29:                $tags_{e'} \leftarrow tags_{e'} \cup sm_i$
30:                $weight(e') \leftarrow w_l - |tags_{e'}|/(i + 1)$
31:            **else**
32:                add the link $e'$ from $u'$ to $v'$ with $l_{e'} = l_e$ to $G$
33:                $tags_{e'} \leftarrow sm_i$
34:                $weight(e') \leftarrow w_l$
35:            **end if**
36:        **end for**
37:    **end for**

38: **end function**

---

3. [lines 14–23]: For each link $e = (u, v)$ in $sm(s_i)$ where $v$ is a data node, we search the graph to see if there is a match for this pattern. We first use $H$ to find the node $u'$ in $G$ to which the node $u$ is mapped. If $u'$ does not have any outgoing link with a label

equal to the label of $e$, we add a new data node $v'$ to $G$. We add $v$ and its mapped data node $v'$ to $H$.

4. [lines 24–37]: For each link $e = (u, v)$ in $sm(s_i)$, we find the nodes in $G$ to which $u$ and $v$ are mapped (say $u'$ and $v'$). If $G$ includes a link with the same label as the label of $e$ between $u'$ and $v'$, we only add $s_i$ to the tags associated with the link. Otherwise, we add a new link to the graph and tag it with $s_i$.

**Adding semantic types**: Once the known semantic models are added to $G$, we add the semantic types learned for the attributes of the target source. As mentioned before, we have two kinds of semantic types: $\langle class\_uri \rangle$ for attributes whose data values are URIs and $\langle class\_uri, property\_uri \rangle$ for attributes that have literal data. For each learned semantic type $t$, we search the graph to see whether $G$ includes a *match* for $t$.

- $t = \langle class\_uri \rangle$: We say $(u, v, e)$ is a match for $t$ if $u$ is a class node with the label $class\_uri$, $v$ is a data node, and $e$ is a link from $u$ to $v$ with the label *karma:uri*. For example, in Fig. 4, $(n_3, n_9, karma:uri)$ is a match for the semantic type $\langle edm:WebResource \rangle$.
- $t = \langle class\_uri, property\_uri \rangle$: We say $(u, v, e)$ is a match for $t$ if $u$ is a class node labeled with $class\_uri$, $v$ is a data node, and $e$ is a link from $u$ to $v$ labeled with $property\_uri$. In Fig. 4, $(n_6, n_{10}, skos:prefLabel)$ is a match for the semantic type $\langle skos:Concept, skos:prefLabel \rangle$.

We say $t = \langle class\_uri \rangle$ or $t = \langle class\_uri, property\_uri \rangle$ has a *partial match* in $G$ when we cannot find a full match for $t$ but there is a class node in $G$ whose label matches $class\_uri$. For instance, the semantic type $\langle skos:Concept, rdfs:label \rangle$ only has a partial match in $G$, because $G$ contains a class node labeled with $skos:Concept$ ($n_6$), but this class node does not have an outgoing link with the label $rdfs:label$.

Algorithm 3 shows the function that adds the learned semantic types to the graph $G$. For each semantic type $t$ learned in the labeling step, we add the necessary nodes and links to $G$ to create a match or complete existing partial matches. Consider the semantic types learned for the source *dia* (Table 1). Fig. 5 illustrates the graph $G$ after adding the semantic types. The nodes and the links that are added in this step are depicted with the blue color. For $\langle aac:CulturalHeritageObject, dcterms:title \rangle$, we do not need to change $G$, because the graph already contained one match: $(n_2, n_5, dcterms:title)$. The semantic type $\langle skos:Concept, rdfs:label \rangle$ only had one partial match ($n_6$), thus, we add one data node ($n_{18}$ with a label equal to the name of the corresponding attribute) and one link (*rdfs:label* from $n_6$ to $n_{18}$) in order to complete the existing
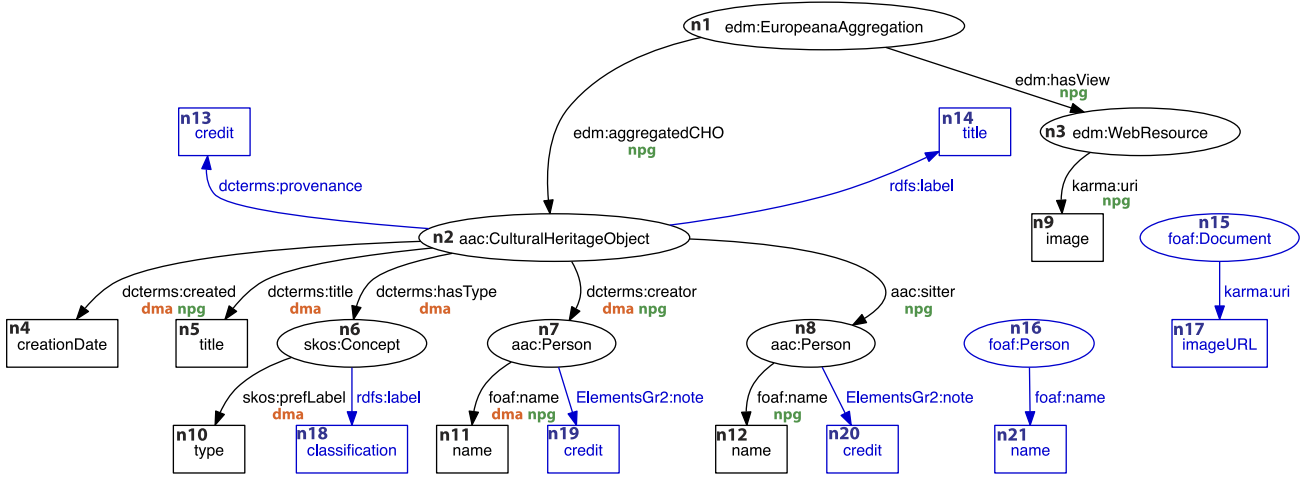
**Fig. 5.** The graph $G$ after adding the nodes and the links corresponding to the semantic types (shown in blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

---

**Algorithm 3** *Add Semantic Types to G*

1: **function** ADDSEMANTICTYPES($G,T$)

2:     **for** each $a_i \in attributes(s)$ **do**
3:         **for** each $t_{ij}^{p_{ij}} \in (t_{i1}^{p_{i1}}, \ldots, t_{ik}^{p_{ik}})$ **do**
4:             **if** $t_{ij} = \langle class\_uri \rangle$ **then**
5:                 $l_v \leftarrow class\_uri$
6:                 $l_e \leftarrow$ "karma: uri"
7:             **else if** $t_{ij} = \langle class\_uri, property\_uri \rangle$ **then**
8:                 $l_v \leftarrow class\_uri$
9:                 $l_e \leftarrow property\_uri$
10:             **end if**
11:             **if** no node in $G$ has the label $l_v$ **then**
12:                 add a new node $v$ with the label $l_v$ in $G$
13:             **end if**
14:             $V_{match} \leftarrow$ all the class nodes with the label $l_v$
15:             $w_h \leftarrow |E|$
16:             **for** each $v \in V_{match}$ **do**
17:                 **if** $v$ does not have an outgoing link labeled $l_e$ **then**
18:                     add a data node $w$ with the label $a_i$ to $G$
19:                     add a link $e = (v, w)$ with the label $l_e$
20:                     $weight(e) \leftarrow w_h$
21:                 **end if**
22:             **end for**
23:         **end for**
24:     **end for**

25: **end function**

partial match. The semantic type $\langle foaf:Document \rangle$ had neither a match nor a partial match. We add a class node ($n_{15}$), a data node ($n_{17}$), and a link between them (*karma:uri* from $n_{15}$ to $n_{17}$) to create a match.

**Adding paths from the ontology**: We use the domain ontology to find all the paths that relate the current class nodes in $G$ (Algorithm 4). The goal is to connect class nodes of $G$ using the direct paths or the paths inferred through the subclass hierarchy in $O$. The final graph is shown in Fig. 6. We connect two class nodes in the graph if there is an object property or *subClassOf* relationship that connects their corresponding classes in the ontology. For instance, in Fig. 6, there is the link *ore:aggregates* from $n_1$ to $n_2$. This link is added because the object property *ore:aggregates* is defined with *ore:Aggregation* as domain and *ore:AggregatedResource* as range, and *edm:EuropeanaAggregation* is a subclass of the class *ore:Aggregation* and *aac:CulturalHeritageObject* is a subclass of *edm:ProvidedCHO*, which is in turn a subclass of the class *ore:AggregatedResource*. As another example, the reason why $n_1$ is connected to $n_{15}$ is that the property *foaf:page* is defined from *owl:Thing* to *foaf:Document* in the FOAF ontology. Thus, a link with

---

**Algorithm 4** *Add Ontology Paths to G*

1: **function** ADDONTOLOGYPATHS($G,O$)

2:     **for** each pair of class nodes $u$ and $v$ in $G$ **do**
3:         $c_1 \leftarrow$ ontology class with $uri = l_u$
4:         $c_2 \leftarrow$ ontology class with $uri = l_v$
5:         $P_{(c_1,c_2)} \leftarrow$ all the direct and inferred properties (including *rdfs:subClassOf*) from $c_1$ to $c_2$ in $O$
6:         $w_h \leftarrow |E|$
7:         **for** each property $p \in P$ **do**
8:             $l_e \leftarrow uri$ of the property $p$
9:             **if** there is no link with label $l_e$ from $u$ to $v$ **then**
10:                 add a link $e = (u, v)$ with label $l_e$ to $G$
11:                 $weight(e) \leftarrow w_h$
12:             **end if**
13:         **end for**
14:     **end for**

15: **end function**

the label *foaf:page* would exist from each class node in $G$ to $n_{15}$ since all classes are subclasses of the class *owl:Thing*. Depending on the size of the ontology, many nodes and links may be added to the graph in this step. To make the figure readable, only a few of the added nodes and links are illustrated in Fig. 6 (the ones with the red color).

In cases where $G$ consists of disconnected components, we add a class node with the label *owl:Thing* to the graph and connect the class nodes that do not have any parent to this root node using a *rdfs:subClassOf* link. This converts the original graph to a graph with only one connected component.

The links in the graph $G$ are weighted. Assigning weights to the links of the graph is very important in our algorithm. We can divide the links in $G$ into two categories. The first category includes the links that are associated with the known semantic models (black links in Fig. 6). The other group consists of the links added from the learned semantic types or the ontology (blue and red links) which are not tagged with any identifier. The basis of our weighting function is to assign a much lower weight to the links in the former group compared to the links in the latter group. If $w_l$ is the default weigh of a link in the first group and $w_h$ is the default weight of a link in the second group, we will have $w_l \ll w_h$. The intuition behind this decision is to produce more coherent models in the next step when we are generating minimum-cost semantic models (Section 3.4). Our goal is to give more priority to the models containing larger segments from the known patterns. One reasonable value for $w_h$ is $w_l * |E|$ in which $|E|$ is the number of links in $G$. This formula ensures that even a long pattern from a
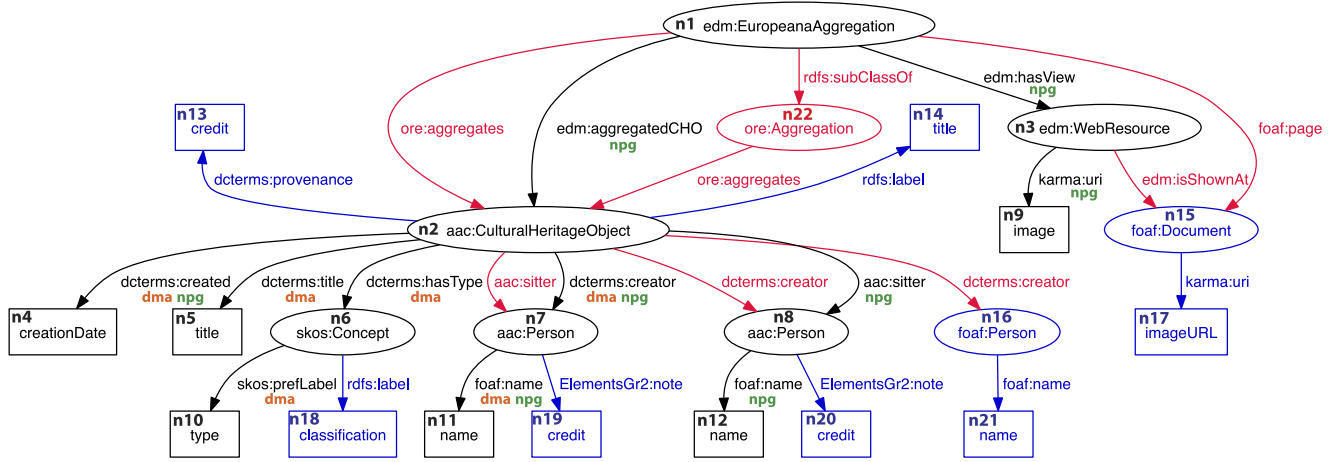
**Fig. 6.** The final graph $G$ after adding the paths from the domain ontologies. For legibility, only a few of all the possible paths between the class nodes are shown (drawn with the red color). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

known semantic model will cost less than a single link that does not exist in any known semantic model.

One factor that we consider in weighting the links coming from the known semantic models (black links) is the popularity of the links, i.e., the number of known semantic models supporting that link. We assign $(w_l - x/(n + 1))$ to each black link where $n$ is the number of known semantic models and $x$ is the number of identifiers the link is tagged with. Suppose that we use $w_l = 1$ in our example. Since our graph in Fig. 6 has a total of 26 links, we will have $w_h = w_l * |E| = 26$. In Fig. 6, the link *edm:hasView* from $n_1$ to $n_3$ will be weighted with 0.66 because it is only supported by $sm(npg)$ ($n = 2, x = 1$). The weight of the link *dcterms:creator* from $n_2$ to $n_7$ will be 0.33 since both $sm(dma)$ and $sm(npg)$ contain that link (the link has two tags).

We assign $w_h$ to the links that are not associated with the known models (blue and red links, which do not have a tag). There is only a small adjustment for the links coming from the ontology (red links). We prioritize direct properties over inherited properties by assigning a slightly higher weight ($w_h + \epsilon$) to the inherited ones. The rationale behind this decision comes from this observation that the direct properties (more specific) are more likely to be used in the semantic models than the inherited properties (more general). For instance, the red link *aac:sitter* from $n_2$ to $n_7$ will be weighted with $w_h = 26$, because its definition in the ontology AAC has *aac:CulturalHeritageObject* as domain and *aac:Person* as range. In other hand, the weight of the link *ore:aggregates* from $n_1$ to $n_2$ will be 26.01 (assume $\epsilon = 0.01$) since the domain of *ore:aggregates* in the ontology ORE is the class *ore:Aggregation* (which is a superclass of *edm:Europeana Aggregation*) and its range is the class *ore:AggregatedResource* (which is a superclass of *aac:CulturalHeritageObject*).

### 3.3. Mapping source attributes to the graph

We use the graph built in the previous step to infer the relationships between the source attributes. First, we find mappings from the source attributes to a subset of the nodes of the graph. Then, we use these mappings to generate and rank candidate semantic models. In this section, we describe the mapping process, and in Section 3.4, we talk about computing candidate semantic models.

To map the attributes of a source to the nodes of $G$ (Fig. 6), we search $G$ to find the nodes matching the semantic types associated with the attributes. For example, the attribute *classification* in *dia* maps to $\{n_6, n_{10}\}$ and $\{n_6, n_{18}\}$, corresponding to the semantic types ⟨*skos:Concept*,*skos:prefLabel*⟩ and ⟨*skos:Concept*,*rdfs:label*⟩, respectively.

Since each attribute has been annotated with $k$ semantic types and also each semantic type may have more than one match in $G$ (e.g., ⟨*aac:Person*,*foaf:name*⟩ maps to $\{n_7, n_{11}\}$ and $\{n_8, n_{12}\}$), more than one mapping $m$ might exist from the source attributes to the nodes of $G$. Generating all the mappings is not feasible in cases where we have a data source with many attributes and the learned semantic types have many matches in the graph. The problem becomes worse when we generate more than one candidate semantic type for each attribute. Suppose that we are modeling the source $s$ consisting of $n$ attributes and we have generated $k$ semantic types for each attribute. If there are $r$ matches for each semantic type, we will have $(k * r)^n$ mappings from the source attributes to the nodes of $G$.

We present a heuristic search algorithm that explores the space of possible mappings as we map the semantic types to the nodes of the graph and expands only the most promising mappings. The algorithm scores the mappings after processing each attribute and removes the low score ones. Our scoring function takes into account the confidence values of the semantic types, the coherence of the nodes in the mappings, and the size of the mappings. The inputs to the algorithm are the learned semantic types $T = \{(t_{11}^{p_{11}}, \ldots, t_{1k}^{p_{1k}}), \ldots, (t_{m1}^{p_{m1}}, \ldots, t_{mk}^{p_{mk}})\}$ for the attributes of the source $s(a_1, a_2, \ldots, a_m)$ and the graph $G$, and the output is a set of candidate mappings $m$ from the source attributes to a subset of the nodes in $G$. The key idea is that instead of generating all the mappings (which is not feasible), we score the partial mappings after processing each attribute and prune the mappings with lower scores. In other words, as soon as we find the matches for the semantic types of an attribute, we rank the partial mappings and keep the better ones. In this way, the number of candidate mappings never exceeds a fixed size (*branching factor*) after mapping each attribute.

Algorithm 5 shows our mapping process. The heart of the algorithm is the scoring function we use to rank the partial mappings (line 22 in Algorithm 5). We compute three functions for each mapping $m$: *confidence(m)*, *coherence(m)*, and *sizeReduction(m)*. Then, we calculate the final score *score(m)* by combining the values of these three functions. We explain these functions using an example. Suppose that the maximum number of the mappings we expand in each step is 2 (*branching_factor* = 2). After mapping the second attribute of the source *dia* (*credit*), we will have: *mappings* = {

    $m_1$: {*title, credit*} → {$(n_2, n_5), (n_2, n_{13})$},
    $m_2$: {*title, credit*} → {$(n_2, n_5), (n_7, n_{19})$},
    $m_3$: {*title, credit*} → {$(n_2, n_5), (n_8, n_{20})$},
    $m_4$: {*title, credit*} → {$(n_2, n_{14}), (n_2, n_{13})$},

**Algorithm 5** *Generate Candidate Mappings*

> **Input:**
>    - $G(V, E)$,
>    - $attributes(s) = \{a_1, \ldots, a_m\}$
>    - $T = \{(t_{11}^{p_{11}}, \ldots, t_{1k}^{p_{1k}}), \ldots, (t_{m1}^{p_{m1}}, \ldots, t_{mk}^{p_{mk}})\}$
>    - *branching_factor*: max number of mappings to expand
>    - *num_of_candidates*: number of candidate mappings
> **Output:** a set of candidate mappings $m$ from $attributes(s)$ to $S \subset V$

```
1:  mappings ← {}
2:  candidates ← {}
3:  for each aᵢ ∈ attributes(s) do
4:      for each t_ij^{pij} ∈ (t_i1^{pi1}, ..., t_ik^{pik}) do
5:          matches ← all the (u, v, e) in G matching t_ij
6:          if mappings = {} then
7:              for each (u, v, e) ∈ matches do
8:                  m ← ({aᵢ} → {u, v})
9:                  mappings ← mappings ∪ m
10:             end for
11:         else
12:             for each m : X → Y ∈ mappings do
13:                 for each (u, v, e) ∈ matches do
14:                     m' ← (X ∪ {aᵢ} → Y ∪ {u, v})
15:                     mappings ← mappings ∪ m'
16:                 end for
17:                 remove m from mappings
18:             end for
19:         end if
20:     end for
21:     if |mappings| > branching_factor then
22:         compute score(m) for each m ∈ mappings
23:         sort items in mappings descending based on their score
24:         keep top branching_factor mappings and remove others
25:     end if
26: end for
27: candidates ← top num_of_candidates items from mappings
    return candidates
```

$$m_5: \{title, credit\} \rightarrow \{(n_2, n_{14}), (n_7, n_{19})\},$$
$$m_6: \{title, credit\} \rightarrow \{(n_2, n_{14}), (n_8, n_{20})\}$$
}.

There are two matches for the attribute *title*: $(n_2, n_5)$ for the semantic type ⟨*aac*:*CulturalHeritageObject*,*dcterms*:*title*⟩ and $(n_2, n_{14})$ for the semantic type ⟨*aac*:*CulturalHeritageObject*,*rdfs*:*label*⟩; and three matches for the attribute *credit*: $(n_2, n_{13})$ for the semantic type ⟨*aac*:*CulturalHeritageObject*,*dcterms*:*provenance*⟩ and $(n_7, n_{19})$ and $(n_8, n_{20})$ for the semantic type ⟨*aac*:*Person*, *ElementsGr*2:*note*⟩. This yields $2 * 3 = 6$ different mappings. Since *branching_factor* $= 2$, we have to eliminate four of these mappings. Now, we describe how the algorithm ranks the mappings.

**Confidence**: We define confidence as the arithmetic mean of the confidence values associated with a mapping. For example, $m_1$ is consisting of the matches for the semantic types ⟨*aac*:*CulturalHeritageObject*,*dcterms*:*title*⟩$^{0.49}$ and ⟨*aac*:*Cultural HeritageObject*,*dcterms*:*provenance*⟩$^{0.83}$. Thus, $confidence(m_1) = 0.66$.

**Coherence**: This function measures the largest number of nodes in a mapping that belong to the same known semantic model. Like the links, the nodes in $G$ are also tagged with the model identifiers although we have not shown them in Fig. 6. We calculate coherence as the maximum number of the nodes in a mapping that have at least one common tag. For instance, $coherence(m_1) = 0.66$ because two nodes out of the three nodes in $m_1$ ($n_2$ and $n_5$) are from $sm(dma)$, and $coherence(m_2) = 1.0$ because all the nodes of $m_2$ are from the same semantic model $sm(dma)$. The goal of defining the coherence is to give more priority to the models containing larger segments from the known patterns.

**Size reduction**: We define the size of a mapping $size(m)$ as the number of the nodes in the mapping. Since we prefer concise models, we seek mappings with fewer nodes. If a mapping has $k$ attributes, the smallest possible size for this mapping is $l = k + 1$ (when all the attributes map to the same class node, e.g., $m_1$) and the largest is $u = 2 * k$ (when all the attributes map to different class nodes, e.g., $m_2$). Thus, the possible size reduction in a mapping is $u - l$. We define $sizeReduction(m) = (u - size(m))/(u - l + 1)$ as how much the size of a mapping is reduced compared to the possible size reduction. For example, $sizeReduction(m_1) = 0.5$ and $sizeReduction(m_2) = 0$.

**Score($m$)**: The final score is the combination the values $confidence(m)$, $coherence(m)$, and $sizeReduction(m)$, which are all in the range $[0, 1]$. We assign a weight to each of these values and then compute the final score as the weighted sum of them: $score(m) = w_1\ confidence(m) + w_2\ coherence(m) + w_3\ sizeReduction(m)$, where $w_1$, $w_2$, and $w_3$ are the weights, decimal values in the range $[0, 1]$ summing up to 1. The proper values of the weights can be tuned by experiments. In our evaluation (Section 4), we obtained better results when all the three functions contributed equally to the final score. That is, $score(m)$ is calculated as the arithmetic mean of $confidence(m)$, $coherence(m)$, and $sizeReduction(m)$ ($w_1 = w_2 = w_3 = 1/3$).

In our example, if we use arithmetic mean to compute the final score, the scores of the 6 mappings we mentioned before are as follows: $score(m_1) = 0.60$, $score(m_2) = 0.42$, $score(m_3) = 0.42$, $score(m_4) = 0.46$, $score(m_5) = 0.39$, $score(m_6) = 0.39$. Therefore, $m_2$, $m_3$, $m_5$, and $m_6$ will be removed from the *mappings* (line 24), and the algorithm continues to the next iteration, which is mapping the next attribute of the source *dia* (*classification*) to the graph. At the end, we will have maximum *branching_factor* mappings, each of them will include all the attributes. We sort these mappings based on their score and consider the top *num_of_candidates* mappings as the candidates (Algorithm 5 line 27).

### 3.4. Generating and ranking semantic models

Once we generated candidate mappings from the source attributes to the nodes of the graph, we compute and rank candidate semantic models. To compute a semantic model for a mapping $m$, we find the minimum-cost tree in $G$ that connects the nodes of $m$. The cost of a tree is the sum of the weights on its links. This problem is known as the Steiner Tree problem [26]. Given an edge-weighted graph and a subset of the vertices, called Steiner nodes, the goal is to find the minimum-weight tree that spans all the Steiner nodes. The general Steiner tree problem is NP-complete, however, there are several approximation algorithms [26–29] that can be used to gain a polynomial runtime complexity.

The inputs to the algorithm are the graph $G$ and the nodes of $m$ (as Steiner nodes) and the output is a tree that we consider as a candidate semantic model for the source. For example, for the source *dia* and the mapping $m$: {*title*, *credit*, *classification*, *name*, *imageURL*} $\rightarrow$ {$(n_2, n_5)$, $(n_2, n_{13})$, $(n_6, n_{10})$, $(n_7, n_{11})$, $(n_3, n_9)$}, the resulting Steiner tree will be exactly as what is shown in Fig. 3(c), which is the correct semantic model of the source *dia*. The algorithm to compute the minimal tree prefers the links that appear in the known semantic models (links with tags) because they have a much lower weight than the other links in $G$. Additionally, since the weight of a link with tags has inverse relation with its number of tags (number of known semantic models containing the link), the semantic model obtained by computing the minimal tree will contain the links that are more popular in the known semantic models.

Selecting more popular links does not always yield the correct semantic model. Suppose that we have three known semantic models {$sm(s_1)$, $sm(s_2)$, $sm(s_3)$}. One of them connects *aac*:*CulturalHeritageObject* to two instances of *aac*:*Person* using
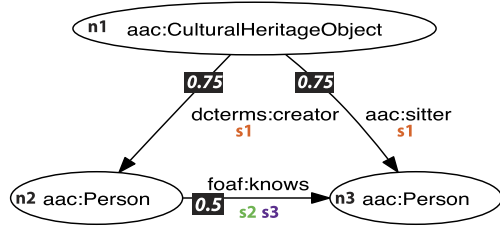
**Fig. 7.** A small part of an example graph constructed using three known models.

the links *dcterms:creator* and *aac:sitter* (similar to *sm(npg)*). The other two semantic models do not contain the class node *aac:CulturalHeritageObject*, but they have two class nodes *aac:Person* connected using the link *foaf:knows*. Fig. 7 shows a small part of the graph constructed using these known models. The black labels on the links represent the weights of the links. For instance, the link *dcterms:creator* from $n_1$ to $n_2$ has a weight equal to 0.75 because it is only supported by $sm(s_1)$ ($w_l - x/(n+1) = 1 - 1/(1+3) = 0.75$).

Now, assume that we have a new source $s_4$ with three attributes $\{a_1, a_2, a_3\}$ annotated with *aac:CulturalHeritageObject*, *aac:Person*, and *aac:Person*. Computing the minimal tree for the mapping *m*: $\{a_1, a_2, a_3\} \rightarrow \{n_1, n_2, n_3\}$ will result a tree that consists of the link *foaf:knows* between $n_2$ to $n_3$ and either *dcterms:creator* from $n_1$ to $n_2$ or *aac:sitter* between $n_1$ and $n_3$. Nonetheless, this is not the correct semantic model for the source. When $s_4$ includes *aac:CulturalHeritageObject* in addition to those two *aac:Person*, it is more likely that the source is describing the relations between the cultural heritage objects and the people and not the relations between the people.

We solve this problem by taking into account the coherence of the patterns. Instead of just the minimal Steiner tree, we compute the *top-k* Steiner trees and rank them first based on the coherence of their links and then their cost. In the example shown in Fig. 7, the top-3 results assuming $n_1$, $n_2$, and $n_3$ as the Steiner nodes are:

$T_1 = \{(n_1, n_3, aac:sitter), (n_2, n_3, foaf:knows)\}$
$T_2 = \{(n_1, n_2, dcterms:creator), (n_2, n_3, foaf:knows)\}$
$T_3 = \{(n_1, n_2, dcterms:creator), (n_1, n_3, aac:sitter)\}$

where $cost(T_1) = cost(T_2) = 1.25$ and $cost(T_3) = 1.5$. Once we computed the *top-k* trees, we sort them according to their coherence. The coherence here means the percentage of the links in the Steiner tree that are supported by the same semantic model. It is computed similar to the coherence of the nodes with the difference that we use the tags on the links instead of the tags on the nodes. In our example, the coherence of $T_1$ and $T_2$ will be 0.5 because their links do not belong to the same known semantic model, and the coherence of $T_3$ will be 1.0 since both of its links are tagged with $s_1$. Therefore, $T_3$ will be ranked higher than $T_1$ and $T_2$, although it has higher cost than $T_1$ and $T_2$.

We use a customized version of the BANKS algorithms [30] to compute the *top-k* Steiner trees. The original BANKS algorithm is developed for the problem of the keyword-based search in relational databases, and because it makes specific assumptions about the topology of the graph, applying it directly to our problem eliminates some of the trees from the results. For instance, if two nodes are connected using two links with different weights, it only considers the one with the lower weight and it never generates a tree including the link with the higher weight. We customized the original algorithm to support more general cases.

The BANKS algorithm creates one iterator for each of the nodes corresponding to the semantic types, and then the iterators follow the incoming links to reach a common ancestor. The algorithm uses the iterator's distance to its starting point to decide which link should be followed next. Because our weights have an inverse relation with their popularity, the algorithm prefers more frequent

links. To make the algorithm converge to more coherent models first, we use a heuristic that prefers the links that are parts of the same pattern (known semantic model) even if they have higher weights. Suppose that $sm_1$: $v_1 \xrightarrow{e_1} v_2$ and $sm_2$: $v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} v_3$ are the only known models used to build the graph *G*, and the weight of the link $e_2$ is higher than $e_1$. Assume that $v_1$ and $v_3$ are the semantic labels. The algorithm creates two iterators, one starting from $v_1$ and one from $v_3$. The iterator that starts from $v_3$ reaches $v_2$ by following the incoming link $v_2 \xrightarrow{e_3} v_3$. At this point, it analyzes the incoming links of $v_2$ and although $e_1$ has lower weight than $e_2$, it first chooses $e_2$ to traverse next. This is because $e_2$ is part of the known model $sm_2$ which includes the previously traversed link $e_3$.

It is important to note that considering coherence of patterns in scoring the mappings and also ranking the final semantic models enables our approach to compute the correct semantic model in many cases where the top semantic types are not the correct ones. For example, for the source *dia*, the mapping *m*:{*title*, *credit*, *classification*, *name*, *imageURL*} $\rightarrow$ {$(n_2, n_5), (n_2, n_{13}), (n_6, n_{10}), (n_7, n_{11}), (n_3, n_9)$}, which maps the attribute *imageURL* to $(n_3, n_9)$ using the type ⟨*edm:WebResource*⟩, will be scored higher than the mapping *m'*:{*title*, *credit*, *classification*, *name*, *imageURL*} $\rightarrow$ {$(n_2, n_5), (n_2, n_{13}), (n_6, n_{10}), (n_7, n_{11}), (n_{15}, n_{17})$}, which maps *imageURL* to $(n_{15}, n_{17})$ using the type ⟨*foaf:Document*⟩. The mapping *m* has lower confidence value than *m'*, but is scored higher because its coherence value is higher. The model computed from the mapping *m* will also be ranked higher than the model computed from *m'*, because it includes more links from known patterns, thus resulting in a lower cost tree.

## 4. Evaluation

We evaluated our approach on two datasets, each including a set of data sources and a set of domain ontologies that will be used to model the sources. Both of these datasets have the same set of data sources, 29 museum sources in CSV, XML, or JSON format containing data from different art museums in the US, however, they include different domain ontologies. The goal is to learn the semantic models of the data sources with respect to two well-known data models in the museum domain: Europeana Data Model (EDM),[16] and CIDOC Conceptual Reference Model (CIDOC-CRM).[17] These data models use different domain ontologies to represent knowledge in the museum domain.

The first dataset, $ds_{edm}$, contains the EDM, AAC, SKOS, Dublin Core Metadata Terms, FRBR, FOAF, ORE, and ElementsGr2 ontologies, and the second dataset, $ds_{crm}$, includes the CIDOC-CRM and SKOS ontologies. The reason why we used two data models is to evaluate how our approach performs with respect to different representations of knowledge in a domain. We applied our approach on both datasets to find the candidate semantic models for each source and then compared the best suggested models (the first ranked models) with models created manually by domain experts. Table 2 shows more details of the evaluation datasets. The datasets including the sources, the domain ontologies, and the gold standard models are available on GitHub.[18] The source code of our approach is integrated into Karma which is available as open source.[19]

Manually constructing semantic models, in addition to being time-consuming and error-prone, requires a thorough understanding of the domain ontologies. Karma [20] provides a user friendly

**Table 2**
The evaluation datasets $ds_{edm}$ and $ds_{crm}$.

| | $ds_{edm}$ | $ds_{crm}$ |
|---|---|---|
| #data source | 29 | 29 |
| #classes in the domain ontologies | 119 | 147 |
| #properties in the domain ontologies | 351 | 409 |
| #nodes in the gold-standard models | 473 | 812 |
| #data nodes in the gold-standard models | 331 | 418 |
| #class nodes in the gold-standard models | 142 | 394 |
| #links in the gold-standard models | 444 | 785 |

graphical interface enabling users to interactively build the semantic models. Yet, building the models in Karma without any automation requires significant user effort. Our automatic approach learns accurate semantic models that can be transformed to the gold standard models by only a few user actions.

In each dataset, we applied our method to learn a semantic model for a target source $s_i$, $sm(s_i)$, assuming that the semantic models of the other sources are known. To investigate how the number of the known models influences the results, we used variable number of known models as input. Suppose that $M_j$ is a set of known semantic models including $j$ models. Running the experiment with $M_0$ means that we do not use any knowledge other than the domain ontology and running it with $M_{28}$ means that the semantic models of all the other sources are known ($M_{28}$ is leave-one-out cross validation). For example, for $s_1$, we ran the code 29 times using $M_0 = \{\}$, $M_1 = \{sm(s_2)\}$, $M_2 = \{sm(s_2), sm(s_3)\}$, ..., $M_{28} = \{sm(s_2), \ldots, sm(s_{29})\}$.

In learning the semantic types of a source $s_i$, we use the data of the sources whose semantic models are known as training data. More precisely, when we are running our labeling algorithm on source $s_i$ with $M_j$ setting, the training data is the data of all the sources $\{s_k | k = 1, \ldots, j \text{ and } k \mathrel{!}= i\}$ and the test data is the data of the target source $s_i$. Using $M_0$ means that there is no training data and thus the labeling function will not be able to suggest any semantic type for the source attributes. To evaluate the labeling algorithm, we use *mean reciprocal rank* (MRR) [31], which is useful when we consider top $k$ semantic types. MRR helps to analyze the ranking of predictions made by any semantic labeling approach using a single measure rather than having to analyze top-1 to top-$k$ prediction accuracies separately, which is a cumbersome task. In learning the semantic types of a source $s_i$ with $n$ attributes, MRR is computed as:

$$MRR = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{rank_i}$$

where $rank_i$ is the rank of the correct semantic type in the top $k$ predictions made for the attribute $a_i$. It is obvious that if we only consider the top semantic type predictions, the value of MRR is equal to the accuracy. In our example in Table 1, MRR $= 1/5(1/1 + 1/1 + 1/1 + 1/1 + 1/2) = 0.9$ (the correct semantic type for the attribute *imageURL* is ranked second).

We compute the accuracy of the learned semantic models by comparing them with the gold standard models in terms of *precision* and *recall*. Assuming that the correct semantic model of the source $s$ is $sm$ and the semantic model learned by our approach is $sm'$, we define precision and recall as:

$$precision = \frac{|rel(sm) \cap rel(sm')|}{|rel(sm')|}$$

$$recall = \frac{|rel(sm) \cap rel(sm')|}{|rel(sm)|}$$

where $rel(sm)$ is the set of triples $(u, v, e)$ in which $e$ is a link from the node $u$ to the node $v$ in the semantic model $sm$. For example, for the semantic model in Fig. 3(c), $rel(sm) =$
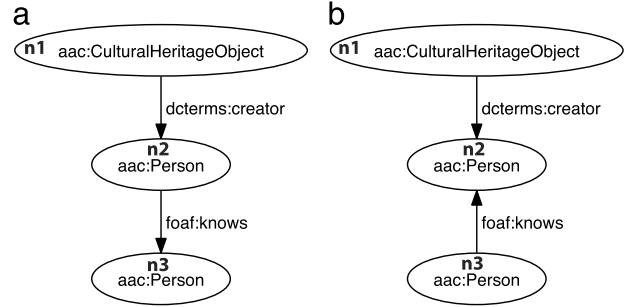


**Fig. 8.** These two semantic models are not equivalent.

$\{(edm{:}EuropeanaAggregation, aac{:}CulturalHeritageObject, edm{:}aggregatedCHO), (edm{:}EuropeanaAggregation, edm{:}WebResource, edm{:}hasView), (aac{:}CulturalHeritageObject, aac{:}Person, dcterms{:}creator), \cdots\}$.

If all the nodes in $sm$ have unique labels and all the nodes in $sm'$ also have unique labels, $rel(sm) = rel(sm')$ ensures that $sm$ and $sm'$ are equivalent. However, if the semantic models have more than one instance of an ontology class, we will have nodes with the same label. In this case, $rel(sm) = rel(sm')$ does not guarantee $sm = sm'$. For example, the two semantic models exemplified in Fig. 8 have the same set of triples although they do not convey the same semantics. In Fig. 8(a), the creator of the artwork knows another person while the semantic model in Fig. 8(b) states that the creator of the artwork is known by another person. Many sources in our datasets have models that include two or more instances of an ontology class.

To have a more accurate evaluation, we number the nodes and then use the numbered labels in measuring the precision and recall. Assume that the model in Fig. 8(a) is the correct semantic model ($sm$) and the one in 8(b) is the model learned by our approach ($sm'$). We change the labels of the nodes $n_1$, $n_2$ and $n_3$ in $sm$ to $aac{:}CulturalHeritageObject1$, $aac{:}Person1$ and $aac{:}Person2$. After this change, we will have $rel(sm) = \{(aac{:}CulturalHeritageObject1, aac{:}Person1, dcterms{:}creator), (aac{:}Person1, aac{:}Person2, foaf{:}knows)\}$. Then, we try all the permutations of the numbering in the learned model $sm'$ and report the precision and recall of the one that generates the best *F1-measure*.[20] For instance, if we number the nodes $n_2$ and $n_3$ in $sm'$ with $aac{:}Person1$ and $aac{:}Person2$, we will have $rel(sm') = \{(aac{:}CulturalHeritageObject1, aac{:}Person1, dcterms{:}creator), (aac{:}Person2, aac{:}Person1, foaf{:}knows)\}$, which yields $precision = recall = 0.5$. If we label $n_2$ with $aac{:}Person2$ and $n_3$ with $aac{:}Person1$, we will have $rel(sm') = \{(aac{:}CulturalHeritageObject1, aac{:}Person2, dcterms{:}creator), (aac{:}Person1, aac{:}Person2, foaf{:}knows)\}$, which still has $precision = recall = 0.5$.

One of the factors influencing the results of our method is the overlap between the known semantic models and the semantic model of the target source. To see how much two semantic models overlap each other, we define the *overlap* metric as the Jaccard similarity between their relationships:

$$overlap = \frac{|rel(sm) \cap rel(sm')|}{|rel(sm) \cup rel(sm')|}.$$

Table 3 reports the minimum, maximum, median, and average overlap between the semantic models of each dataset. Overall, the higher the overlap between the known semantic models and the semantic model of the target source, the more accurate models can be learned.

---

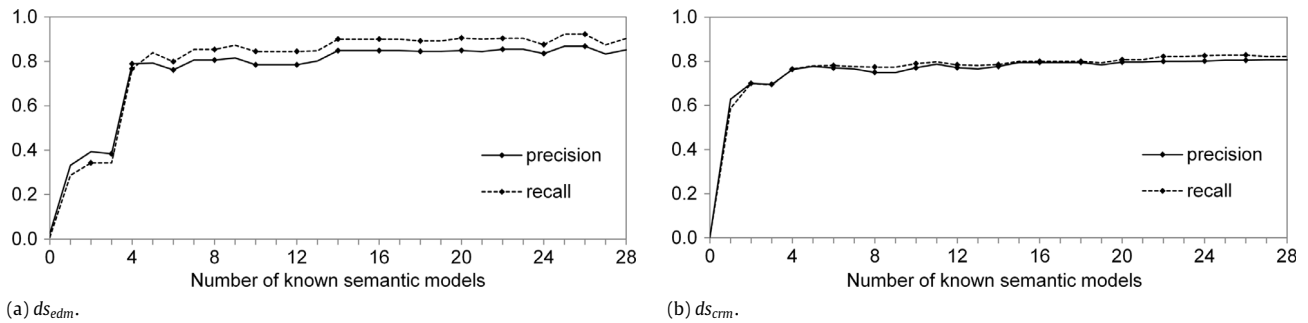20 $F1 - measure = 2 * (precision \times recall)/(precision + recall)$.

(a) $ds_{edm}$.

(b) $ds_{crm}$.

**Fig. 9.** Average precision and recall for the learned semantic models when the attributes are labeled with their correct semantic types.
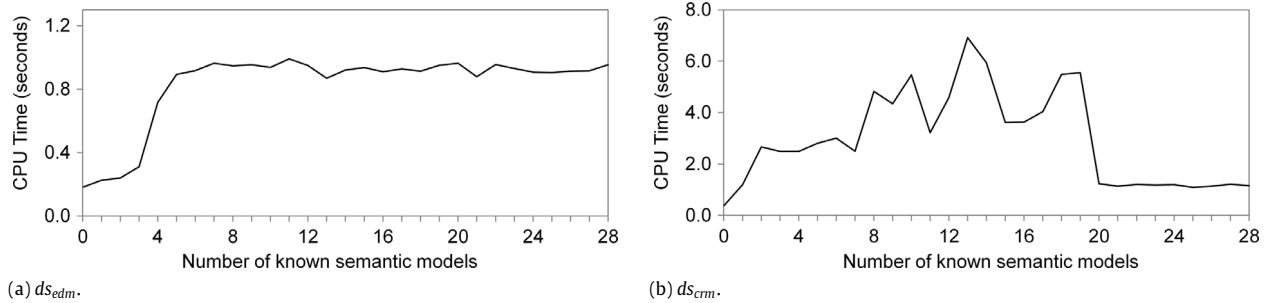


(a) $ds_{edm}$.

(b) $ds_{crm}$.

**Fig. 10.** Average semantic model learning time when the attributes are labeled with their correct semantic types.

**Table 3**
The overlap between the pairs of the semantic models in the datasets $ds_{edm}$ and $ds_{crm}$.

|  | $ds_{edm}$ | $ds_{crm}$ |
|---|---|---|
| Minimum overlap | 0.04 | 0.03 |
| Maximum overlap | 1 | 1 |
| Median overlap | 0.45 | 0.46 |
| Average overlap | 0.43 | 0.46 |

In our mapping algorithm (Algorithm 5), we used 50 as cut-off ($branching\_factor = 50$) and then considered all the generated mappings as the candidate mappings ($num\_of\_candidates = 50$). We justify this choice in Section 4.2 by analyzing the impact of the branching factor on the accuracy of the results and the running time of the algorithm. To score a mapping $m$, we assigned equal weights to the functions $confidence(m)$, $coherence(m)$, and $sizeReduction(m)$. We tried different combinations of weights, and although our algorithm generated more precise models for a few sources in some of these weight systems, the average results were better when each of these functions contributed equally to the final score. Once we found the candidate mappings, we generated the top 10 Steiner trees for each of them ($k = 10$ in *top-k* Steiner tree algorithm). Finally, we ranked the candidate semantic models (at most 500) and compared the best one with the correct model of the source. We ran two experiments with different scenarios that will be explained next.

### 4.1. Scenario 1

In the first scenario, we assumed that each source attribute is annotated with its correct semantic type. The goal was to see how well our approach learns the attribute relationships using the correct semantic types. Fig. 9 illustrates the average precision and recall of all the learned semantic models ($sm'(s_1), \ldots, sm'(s_{29})$) for each $M_j$ ($j \in [0..28]$) for each dataset. Since the correct semantic types are given, we excluded their corresponding triples in computing the precision and recall. That is, we compared only the links between the class nodes in the gold standard models with

the links between the class nodes in the learned models. We call such links *internal links*, the links that are established between the class nodes in semantic models. The total number of the links in the dataset $ds_{edm}$ is 444, and 331 of these links corresponds to the source attributes (there are 331 data nodes). Thus, $ds_{edm}$ has 113 internal links ($444 - 331 = 113$). Following the same rationale, $ds_{crm}$ has 367 internal links.

The results show that the precision and recall increase significantly even with a few known semantic models. An interesting observation is that when there is no known semantic model and the only background knowledge is the domain ontology (baseline, $M_0$), the precision and recall are close to 0. This low accuracy comes from the fact that there are multiple links between each pair of class nodes in the graph $G$, and without additional information, we cannot resolve the ambiguity. Although we assign lower weights to direct properties to prioritize them over inherited ones, it cannot help much because for many of the class nodes in the correct models, there is no object property in the ontology that is explicitly defined with the corresponding classes as domain and range. In fact, most of the properties that have been used in the correct models are either inherited properties or defined without a domain or/and range in the ontology.

To evaluate the running time of the approach, we measured the running time of the algorithm starting from building the graph until ranking the results on a single machine with a Mac OS X operating system and a 2.3 GHz Intel Core i7 CPU. Fig. 10 shows the average time (in seconds) of learning the semantic models. The reason why there is some fluctuations in the timing diagram of $ds_{crm}$ (Fig. 10(b)) is related to the topology of the graph built on top of the known models and also the details of our implementation. While one expects to see linear increase in time when the number of known semantic models grows, sometimes adding a new semantic model changes the structure of the graph in a way that the Steiner tree algorithm finds $k$ candidate trees faster.

We believe that the overall time of the process can be further reduced by using parallel programming and some optimizations in the implementation. For example, the graph can be built incrementally. When a new known model is added, we do not need to create the graph from scratch. We just need to merge the new known model to the existing graph and update the links.
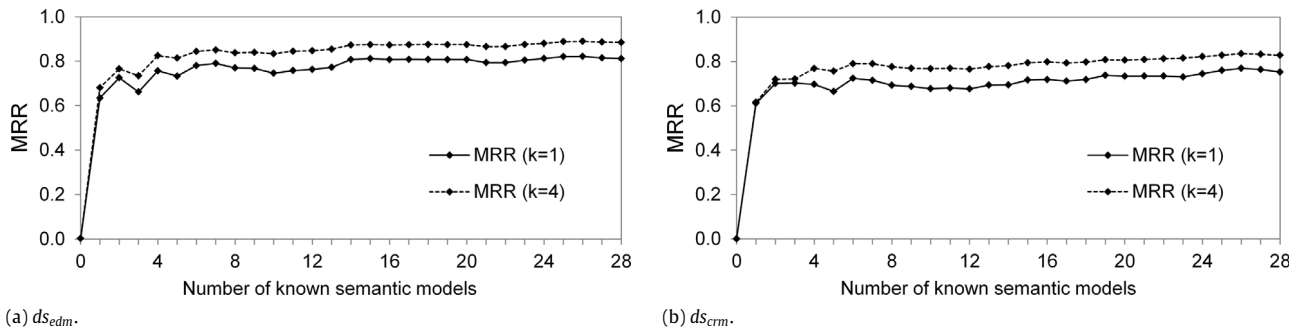
(a) $ds_{edm}$.    (b) $ds_{crm}$.

**Fig. 11.** MRR value of the learned semantic types when only the top learned semantic types are considered ($k = 1$); and the top four suggested types are considered ($k = 4$).
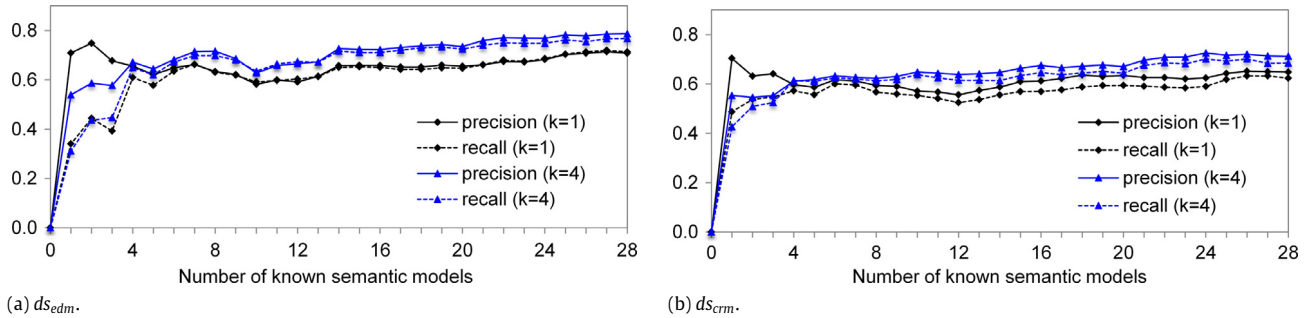


(a) $ds_{edm}$.    (b) $ds_{crm}$.

**Fig. 12.** Average precision and recall for the learned semantic models for $k = 1$ and $k = 4$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

## 4.2. Scenario 2

In the second scenario, we used our semantic labeling algorithm to learn the semantic types. We trained the labeling classifier on the data of the sources whose semantic models are already known and then applied the learned labeling function to the target source to assign a set of candidate semantic types to each source attribute. Fig. 11 shows the MRR diagram for $ds_{edm}$ and $ds_{crm}$ in two cases: (1) only the top semantic type (the type with the highest confidence value) is considered ($k = 1$), (2) the top four learned semantic types are taken into account as the candidate semantic types ($k = 4$). Note that, when $k = 1$, the MRR value is equal to the accuracy, i.e., how many of the attributes are labeled with their correct semantic types.

Once the labeling is done, we feed the learned semantic types to the rest of algorithm to learn a semantic model for each source. The average precision and recall of the learned models are illustrated in Fig. 12. The black color shows the precision and recall for $k = 1$, and the blue color illustrates the precision and recall for $k = 4$. In this experiment, we computed precision and recall for all the links including the links from the class nodes to the data nodes (these links are associated with the learned semantic types). The results show that using the known semantic models as background knowledge yields in a remarkable improvement in both precision and recall compared to the case in which we only consider the domain ontology ($M_0$).

We provide an example to help in understanding the correlation between the MRR and the precision values, i.e., how the accuracy of the learned semantic types affects the accuracy of the learned semantic models. The average MRR value for $ds_{crm}$ when we use $k = 1$ and $M_{28}$ (leave-one-out setting) is 0.75 (Fig. 11(b)). This means that our labeling algorithm can learn the correct semantic types for only 75% of the attributes. From Table 2, we know that the gold standard models for $ds_{crm}$ have totally 418 data nodes, and thus, 418 links in the gold standard models correspond to the source attributes. Since 75% of the attributes are labeled correctly, 313 links out of 418 links corresponding to the source attributes

will be correct in the learned semantic models. Even if we predict all the internal links correct ($785 - 418 = 367$ links), the maximum precision would be 86% (($367 + 313)/785$). However, the input to the Steiner tree algorithm are the nodes coming from the learned semantic types (leaves of the tree), and incorrect semantic types may prompt the Steiner tree algorithm to select incorrect links in the higher levels (internal links). As we see in Fig. 12(b), in the $k = 1$ and $M_{28}$ setting, the average precision of the learned semantic models is 65%.

When considering the top four semantic types ($k = 4$) instead of only the top one semantic type ($k = 1$), our algorithm recovers some of the correct semantic types even if they are not the top predictions of the labeling function. For example, in the dataset $ds_{crm}$, using $k = 4$ rather than $k = 1$ when we have 28 known models ($M_{28}$), improves the precision by 6% and the recall by 7% (Fig. 12(b)). This improvement is mainly because of the coherence factor we take into account in scoring the mappings and also ranking the candidate semantic models.

The running time of the algorithm in the second scenario is displayed in Fig. 13. This time does not include the labeling step. The work done by Krishnamurthy et al. [14] contains a detailed analysis of the performance of the labeling algorithm. As we can see in Fig. 13(b), the running time of the algorithm is higher at $M_8$, $M_9$, $M_{10}$, and $M_{11}$ when $k = 1$. This is because computing top 10 Steiner trees takes longer once we add semantic models of $s_8$, $s_9$, $s_{10}$, and $s_{11}$ to the graph. When adding more semantic models, the algorithm runs faster. For example, the average time at $M_{11}$ is 7.29 s while it is 1.21 s at $M_{12}$. This is the result of a combination of several reasons. First, there is more training data in learning the semantic types of a source $s_i$ at $M_{12}$, and this affects the output of the mapping algorithm (Algorithm 5). Second, the structure of the graph is different at $M_{12}$ and this results in different mappings between the source attributes and the graph. Finally, the new semantic model $sm(s_{12})$ adds new paths to the graph allowing the Steiner tree algorithm to find the top 10 trees faster.

We mentioned earlier that we used 50 as the value of the branching factor in mapping the source attributes to the graph
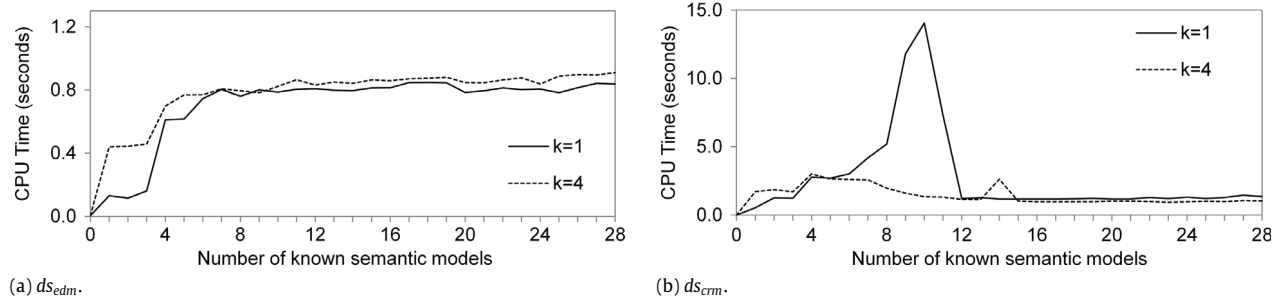
(a) $ds_{edm}$.

(b) $ds_{crm}$.

**Fig. 13.** Average semantic model learning time when the attributes are labeled with their correct semantic types.
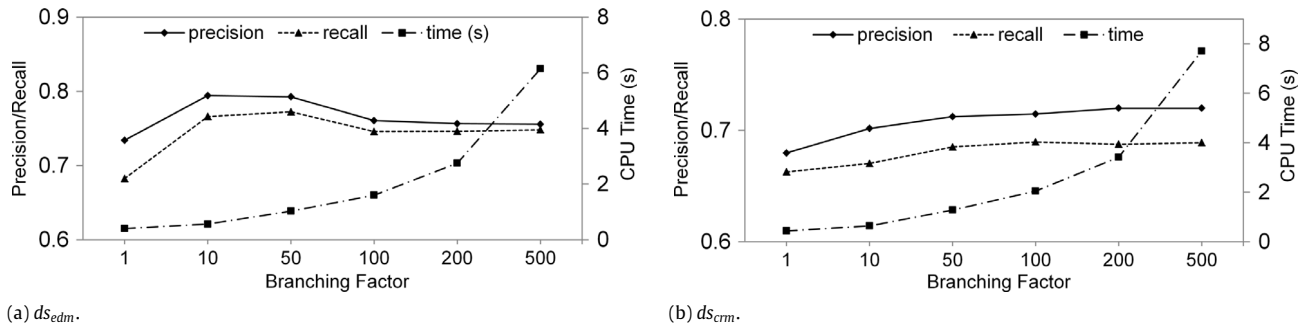


(a) $ds_{edm}$.

(b) $ds_{crm}$.

**Fig. 14.** Impact of branching factor on precision, recall, and running time for $k = 4$ and $M_{28}$.

(line 21 of Algorithm 5). The branching factor is essential to the scalability of our mapping algorithm. This value can be configured by trying some sample values and then choosing a value yielding good accuracy while keeping the running time of the algorithm reasonably low. This can be different for each dataset. In our evaluation, using *branching_factor* $= 50$ worked well for both datasets. Fig. 14 illustrates how changing the value of the branching factor affects the precision, recall, and running time of the algorithm in a setting where we considered 4 candidate semantic types ($k = 4$) and the semantic models of all the other sources were known ($M_{28}$). In this experiment, we fixed the value of *num_of_candidates* (line 27 of Algorithm 5) equal to the value of *branching_factor*. This means that all the generated mappings will be given to the Steiner tree algorithm as the candidate mappings. As we can see in Fig. 14(b), increasing the value of the branching factor from 50 to 200 for $ds_{crm}$ provides 1% improvement in the precision, however, it increases the average running time by 2.14 s. We chose to ignore this insignificant increase in the precision and used 50 as the branching factor to gain a better running time.

## 5. Related work

The problem of describing semantics of data sources is at the core of data integration [1] and exchange [32]. The main approach to reconcile the semantic heterogeneity among sources consists of defining logical mappings between the source schemas and a common target schema. One way to define these mappings is local-as-view (LAV) descriptions where every source is defined as a view over the domain schema [1]. The semantic models that we generate are graphical representation of LAV rules, where the domain schema is the domain ontology. Although the logical mappings are declarative, defining them requires significant technical expertise, so there has been much interest in techniques that facilitate their generation.

In traditional data integration, the mapping generation problem is usually decomposed in a *schema matching* phase followed by *schema mapping* phase [33]. Schema matching [34] finds correspondences between elements of the source and target schemas.

For example, iMAP [35] discovers complex correspondences by using a set of special-purpose searchers, ranging from data overlap, to machine learning and equation discovery techniques. This is analogous to the semantic labeling step in our work [14], where we learn a labeling function to learn candidate semantic types for a source attribute. Every semantic type maps an attribute to an element in the domain ontology (a class or property in the domain ontology).

Schema mapping defines an appropriate transformation that populates the target schema with data from the sources. Mappings may be arbitrary procedures, but of greater interest are declarative mappings expressible as queries in SQL, XQuery, or Datalog. These mapping formulas are generated by taking into account the schema matches and schema constraints. There has been much research in schema mapping, from the seminal work on Clio [36], which provided a practical system and furthered the theoretical foundations of data exchange [37] to more recent systems that support additional schema constraints [38]. Alexe et al. [39] generate schema mappings from examples of source data tuples and the corresponding tuples over the target schema. An et al. [40] generate declarative mapping expressions between two tables with different schemas starting from element correspondences. They create a graph from the conceptual model (CM) of each schema and then suggest plausible mappings by exploring low-cost Steiner trees that connect those nodes in the CM graph that have attributes participating in element correspondences. Their work is similar to our previous semi-automatic approach to build the semantic models [20], where we derive a graph from the domain ontology and the learned semantic types. We exploited the knowledge from the ontology to assign weights to the links based on their types, e.g., direct properties get lower weight than inherited properties, because we wanted to give more priority to more specific relations. We also allow the user to correct the mappings interactively. In the current paper, in addition to the ontology, we consider previous known semantic models to improve the modeling of an unknown source.

Our work on learning semantic models of structured sources is complementary to these schema mapping techniques. Instead of focusing on satisfying schema constraints, we analyze known

source models to propose mappings that capture more closely the semantics of the target source in ways that schema constraints could not disambiguate. For example, by suggesting that a *dcterms:creator* relationship is more likely than *dbpedia:owner* in a given domain. Moreover, our algorithm can incrementally refine the mappings based on user feedback and learn from this feedback to improve future predictions.

In the Semantic Web, what is meant by a source description is a semantic model describing the source in terms of the concepts and relationships defined by a domain ontology. There are many studies on mapping data sources to ontologies. Several approaches have been proposed to generate semantic web data from databases and spreadsheets [5].

D2R [41,42] and D2RQ [43] are mapping languages that enable the user to define mapping rules between tables of relational databases and target ontologies in order to publish semantic data in RDF format. R2RML [19] is a another mapping language, which is a W3C recommendation for expressing customized mappings from relational databases to RDF datasets. Writing the mapping rules by hand is a tedious task. The users need to understand how the source table maps to the target ontology. They also need to learn the syntax of writing the mapping rules. RDOTE [8] is a tool that provides a graphical user interface to facilitate mapping relational databases into ontologies. The developers of RDOTE have said they will incorporate an export/import mechanism for D2RQ compliant mapping files, as well as a query builder graphical user interface to hasten the mapping creation process. RDF123 [2] and XLWrap [4] are other tools to define mappings from spreadsheets to RDF graphs. Although these tools can facilitate the mapping process, the users still need to manually define the mappings between the source and target ontologies.

In recent years, there are some efforts to automatically infer the implicit semantics of tables. Polfliet and Ichise [6] use string similarity between the column names and the names of the properties in the ontology to find a mapping between the table columns and the ontology. Wang et al. [12] detect the header of Web tables and use them along with the values of the rows to map the columns to the attributes of the corresponding entity in a rich and general purpose taxonomy of worldly facts built from a corpus of over one million Web pages and other data. This approach can only deal with the tables containing information of a single entity type.

Limaye et al. [7] used YAGO[21] to annotate web tables and generate binary relationships using machine learning approaches. However, this approach is limited to the labels and relations defined in the YAGO ontology (less than 100 binary relationships). Venetis et al. [11] presented a scalable approach to describe the semantics of tables on the Web. To recover the semantics of tables, they leverage a database of class labels and relationships automatically extracted from the Web. They attach a class label to a column if a sufficient number of the values in the column are identified with that label in the database of class labels, and analogously for binary relationships. Although these approaches are very useful in publishing semantic data from tables, they are limited in learning the semantics relations. Both of these approaches only infer individual binary relationships between pair of columns. They are not able to find the relation between two columns if there is no direct relationship between the values of those columns. Our approach can connect one column to another one through a path in the ontology. For example, suppose that we have a table including two columns *person* and *city*, where the city is the location of the company the person is working for. Our approach can learn a semantic model that

connects the class *Person* to the class *City* through the chain *Person* $\xrightarrow{worksFor}$ *Organization* $\xrightarrow{location}$ *City*.

There is also work that exploits the data available in the Linked Open Data (LOD) cloud to capture the semantics of the tables and publish their data as RDF. Munoz et al. [44] mine RDF triples from the Wikipedia tables by linking the cell values to the resources available in DBPedia [45]. This approach is limited to Wikipedia tables because of its simple linking algorithm. If a cell value contains a hyperlink to a Wikipedia page, the Wikipedia URL maps to a DBpedia entity URI by replacing the namespace http://en.wikipedia.org/wiki/ of the URL with http://dbpedia.org/resource/.

In other work, Mulwad et al. [13] used *Wikitology* [46], an ontology which combines some existing manually built knowledge systems such as DBPedia and Freebase [47], to link cells in a table to Wikipedia entities. They query the background LOD to generate initial lists of candidate classes for column headers and cell values and candidate properties for relations between columns. Then, they use a probabilistic graphical model to find the correlation between the columns headers, cell values, and relation assignments. The quality of the semantic data generated by this category of work is highly dependent to how well the data can be linked to the entities in LOD. While for most popular named entities there are good matches in LOD, many tables contain domain-specific information or numeric values (e.g., temperature and age) that cannot be linked to LOD. Moreover, these approaches are only able to identify individual binary relationships between the columns of a table. However, an integrated semantic model is more than fragments of binary relationships between the columns. In a complete semantic model, the columns may be connected through a path including the nodes that do not correspond to any column in the table.

Parundekar et al. [48] previously developed an approach to automatically generate conjunctive and disjunctive mappings between the ontologies of linked data sources by exploiting existing linked data instances. However, the system does not model arbitrary sources such as we present in this paper. Carman and Knoblock [49] use known source descriptions to learn a semantic description that precisely describes the relationship between the inputs and outputs of a source, expressed as a Datalog rule. However, their approach is limited in that it can only learn sources whose models are subsumed by the models of known sources. That is, the description of a new source is a conjunctive *combination* of known source descriptions. By exploring paths in the domain ontology, in addition to patterns in the known sources, we can hypothesize target mappings that are more general than previous source descriptions or their combinations.

In our earlier Karma work [20], we build a graph from learned semantic types and a domain ontology and use this graph to map a source to the ontology *interactively*. In that work, the system uses the knowledge from the domain ontology to propose models to the user, who can correct them as needed. The system remembers semantic type labels assigned by the user, however, it does not learn from the structure of previously modeled sources.

The most closely related work [16,15] on exploiting known semantic models to learn a model for a new unknown source. However, our previous approach was less scalable. When there are many source attributes, there will be a large number of mappings from the source attributes to the nodes of the graph. Even though we used a beam search algorithm in the mapping step to ameliorate this problem, the graph grows as the number of known semantic models grows, which makes computing the semantic models inefficient. In this paper, we have presented a compact graph structure that merges overlapping segments of the known semantic models. We also use a new algorithm to generate and rank the candidate semantic models. We generate candidate

---

[21] http://www.mpi-inf.mpg.de/yago-naga/yago.

models by computing *top-k* Steiner trees and then rank them based on the coherence of the links. This new approach, in addition to generating more accurate semantic models, significantly improves the running time of the learning process. Integrating our algorithm into Karma, enables the user to refine the automatically learned models resulting in more accurate predictions for future data sources.

In recent years, ontology matching has received much attention in the Semantic Web community [50,51]. Ontology matching (or ontology alignment) finds the correspondence between semantically related entities of different ontologies. This problem is analogous to schema matching in databases. Both schemas and ontologies provide a vocabulary of terms that describe a domain of interest. However, schemas often do not provide explicit semantics for their data. Our work benefits from some of the techniques developed for ontology matching. For example, *instance-based ontology matching* exploits similarities between instances of ontologies in the matching process. Our semantic labeling algorithm adopts the same idea to map the data of a new source to the classes and properties of a target ontology. The algorithm computes the similarity (cosine similarity between TF/IDF vectors) between the data of the new source and the data of the sources whose semantic models are known.

Ontology matching is different than the problem we addressed in this paper in the sense that in our work the data that is being mapped to a target ontology is not bound to any source ontology. This makes our problem more complicated since no explicit semantics is necessarily attached to data sources. Moreover, most of the work on ontology matching only finds simple correspondences such as equivalence and subsumption between ontology classes and properties. Therefore, the explicit relationships within the data elements are often missed in aligning the source data to the target ontology. Suppose that we want to find the correspondences between a source ontology $O_s$ and a target ontology $O_t$. Using ontology matching, we find that the class $A_s$ in $O_s$ maps to the class $A_t$ in $O_t$ and the class $B_s$ in $O_s$ maps to the class $B_t$ in $O_t$. Assume that there is only one property connecting $A_s$ to $B_s$ in $O_s$, but there are multiple paths connecting $A_t$ to $B_t$ in $O_t$. If we align the source data to the target ontology $O_t$ using the correspondences found by ontology matching, the instances of $A_s$ will be mapped to the class $A_t$ and the instances of $B_s$ will be mapped to the class $B_t$. However, this alignment does not tell us which path in $O_t$ captures the correct meaning of the source data.

## 6. Discussion

In this paper, we presented a scalable approach to learn semantic models of structured data sources as mappings from the sources to a domain ontology. Such models are the key ingredients in the process of publishing data into the LOD knowledge graph. The core idea is to exploit the domain ontology and previously learned semantic models to hypothesize a plausible semantic model for a new source. The evaluation shows that our approach learns rich semantic models with minimal user input.

The first step in learning semantic models is learning the semantic types in which the system labels each source attribute with a class or property from the ontology. The output of the labeling step is a set of candidate semantic types and their confidence values rather than one fixed semantic type. Taking into account the uncertainty of the labeling algorithm is very important because machine learning techniques often cannot distinguish the types of the source attributes that have similar data values, e.g., *birthDate* and *deathDate*.

Once the system produces candidate semantic types for each attribute, it creates a graph from known semantic models and augments it by adding the nodes and the links corresponding to the

semantic types and adding the paths inferred from the ontology. The next step is mapping the source attributes to the nodes of the graph where we use a search algorithm that enables the system to do the mapping even when the source has many attributes. The algorithm, after processing each source attribute, prunes the existing mappings by scoring them and removing the ones having lower scores. The proposed scoring function not only contributes to the scalability of our method, but also increases the accuracy of the learned models.

The final part of the approach is computing the minimal tree that connects the nodes of the candidate mappings. This step might be computationally inefficient if we have a very large graph. However, our algorithm to construct the graph consolidates the overlapping segments of the known semantic models, making it scalable to a huge number of known semantic models.

Our learning algorithms play an important role in making the Karma interactive user interface easy to use, a key design goal given that many of our users are domain experts, but are not Semantic Web experts. Our experience observing users is that they can understand and critique models when displayed in our interactive user interface. They can easily verify that models accurately capture the semantics of a source, and can easily spot errors or controversial modeling decisions. Users can click on the corresponding elements on the screen and do local modifications such as replacing the property of a link or changing the source or destination of a link.

We also observe that it is much harder for users to model a source from scratch, as is necessary in tools such as Open Refine.[22] Even though the user interface is easy to use, the task of filling a blank page with a model is daunting for many users. Karma helps these users because it gives them an almost-correct model as a starting point. Users can easily find the elements they do not agree with, and can easily change them. A possible direction for future work is to perform user evaluations to measure the quality of the models produced using learning algorithms. Although time to create models is important, we hypothesize that most users, such as our museum users, are primarily concerned with producing correct models, and time to model is a secondary concern for them. By using previous models, users are more likely to model sources in a correct way.

Our work also plays a role in helping communities to produce consistent Linked Data so that sources containing the same type of data use the same classes and properties when published in RDF. Often, there are multiple correct ways to model the same type of data. For example, users can use Dublin Core and FOAF to model the creator relationship between a person and an object (*dcterms:creator* and *foaf:maker*). A community is better served when all the data with the same semantics is modeled using the same classes and properties. Our work encourages consistency because our learning algorithms bias the selection of classes and properties towards those used more frequently in existing models.

A future direction of our work is to improve the quality of the automatically generated models by leveraging the significant amount of data available in the Linked Open Data (LOD) cloud, which is a vast and growing collection of semantic data that has been published by various data providers. The current estimate is that the LOD cloud contains over 30 billion RDF triples. Even the New York Times is now publishing all of their metadata as Linked Open Data.[23] It should be noted that a nontrivial portion of LOD is just data with limited semantic descriptions, but much of that data has been linked to other sources that does have some form of semantic description. Given the growing availability of this type

---

[22] http://openrefine.org/.

[23] See http://data.nytimes.com.

of data, LOD will provide an invaluable source of semantic content that we can exploit as background knowledge.

Given the huge repository of data available in LOD, for any given set of values provided by a new source, we can search for classes that provide or even subsume all of the data for a given property of a source. For example, if we have a set of values for people names or temperature, we are likely to find some classes in LOD that provides that same set of values. We will not require a perfect overlap between the set of values from the source and a class in the Linked Open Data, but rather a statistically significant overlap, similar to what is done by Parundekar et al. [48]. An important challenge here is how to efficiently find the classes that most closely match the set of attribute values and how to handle the problem that the classes that match the best may come form different ontologies.

We can also exploit LOD to disambiguate the relationships between the attributes [52]. Once we have identified the semantic types of the source attributes, we can search for corresponding classes in LOD and analyze which properties are connecting them. Those properties can be candidates for the relationships between the attributes of the new source. Consider the semantic model of the source *dia* in Fig. 3(c). Once we identify that ⟨*aac:CulturalHeritageObject*,*dcterms:title*⟩ and ⟨*aac:Person*,*foaf:name*⟩ are the semantic types of the first and fourth attributes, we can search LOD for possible properties between instances of the classes *aac:CulturalHeritageObject* and *aac:Person* and find that the properties *dcterms:creator* and *acc:sitter* are better candidates than other properties that ontology suggests, e.g., *dbpedia:owner*. By combining the information we extract for each pair of classes, we can narrow the search to those classes and properties that commonly occur together.

## Acknowledgments

## References

[1] A. Doan, A. Halevy, Z. Ives, Principles of Data Integration, Morgan Kauffman, 2012.
[2] L. Han, T. Finin, C. Parr, J. Sachs, A. Joshi, RDF123: From Spreadsheets to RDF, 2008, pp. 451–466.
[3] A.P. Sheth, K. Gomadam, A. Ranabahu, Semantics enhanced services: METEOR-S, SAWSDL and SA-REST, IEEE Data Eng. Bull. 31 (3) (2008) 8–12.
[4] A. Langegger, W. Wöß, XLWrap—querying and integrating arbitrary spreadsheets with SPARQL, in: A. Bernstein, D.R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), International Semantic Web Conference, in: Lecture Notes in Computer Science, vol. 5823, Springer, 2009, pp. 359–374.
[5] S.S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T.T. Jr., S. Auer, J. Sequeda, A. Ezzat, A survey of current approaches for mapping of relational catabases to RDF (01 2009).
[6] S. Polfliet, R. Ichise, Automated mapping generation for converting databases into linked data, in: A. Polleres, H. Chen (Eds.), ISWC Posters&Demos, in: CEUR Workshop Proceedings, vol. 658, CEUR-WS.org, 2010.
[7] G. Limaye, S. Sarawagi, S. Chakrabarti, Annotating and searching web tables using entities, types and relationships, PVLDB 3 (1) (2010) 1338–1347.
[8] K.N. Vavliakis, T.K. Grollios, P.A. Mitkas, RDOTE—transforming relational databases into semantic web data, in: A. Polleres, H. Chen (Eds.), ISWC Posters & Demos, in: CEUR Workshop Proceedings, vol. 658, CEUR-WS.org, 2010.
[9] L. Ding, D. DiFranzo, A. Graves, J. Michaelis, X. Li, D.L. McGuinness, J.A. Hendler, TWC data-gov corpus: Incrementally generating linked government data from data.gov, in: M. Rappa, P. Jones, J. Freire, S. Chakrabarti (Eds.), WWW, ACM, 2010, pp. 1383–1386.
[10] V. Saquicela, L.M.V. Blázquez, Óscar Corcho, Lightweight semantic annotation of geospatial RESTful services, in: Proceedings of the 8th Extended Semantic Web Conference, ESWC, 2011, pp. 330–344.
[11] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, C. Wu, Recovering semantics of tables on the web, Proc. VLDB Endow. 4 (9) (2011) 528–538.
[12] J. Wang, H. Wang, Z. Wang, K.Q. Zhu, Understanding tables on the web, in: P. Atzeni, D.W. Cheung, S. Ram (Eds.), ER, in: Lecture Notes in Computer Science, vol. 7532, Springer, 2012, pp. 141–155.
[13] V. Mulwad, T. Finin, A. Joshi, Semantic message passing for generating linked data from tables, in: The Semantic Web—ISWC 2013, Springer, 2013, pp. 363–378.
[14] R. Krishnamurthy, A. Mittal, C.A. Knoblock, P. Szekely, Assigning semantic labels to data sources, in: Proceedings of the 12th Extended Semantic Web Conference, ESWC, 2015.
[15] M. Taheriyan, C.A. Knoblock, P. Szekely, J.L. Ambite, A scalable approach to learn semantic models of structured sources, in: Semantic Computing, ICSC, 2014 IEEE International Conference on, 2014, pp. 183–190.
[16] M. Taheriyan, C.A. Knoblock, P. Szekely, J.L. Ambite, A graph-based approach to learn semantic descriptions of data sources, in: Procs. 12th International Semantic Web Conference, ISWC, 2013.
[17] S. Hennicke, M. Olensky, V.D. Boer, A. Isaac, J. Wielemaker, A data model for cross-domain data representation. The Europeana data model in the case of archival and museum data, in: Schriften zur Informationswissenschaft 58, Proceedings des 12. Internationalen Symposiums der Informationswissenschaft, ISI 2011, 2011, pp. 136–147.
[18] M. Doerr, The CIDOC conceptual reference module: An Ontological approach to semantic interoperability of metadata, AI Mag. 24 (3) (2003) 75–92.
[19] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012, http://www.w3.org/TR/r2rml/, 2012.
[20] C. Knoblock, P. Szekely, J.L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, P. Mallick, Semi-automatically mapping structured sources into the semantic web, in: Proc. 9th Extended Semantic Web Conference, 2012.
[21] P. Szekely, C.A. Knoblock, F. Yang, X. Zhu, E. Fink, R. Allen, G. Goodlander, Connecting the Smithsonian American art museum to the linked data cloud, in: Proceedings of the 10th Extended Semantic Web Conference, ESWC, Montpellier, 2013, pp. 593–607.
[22] P. Szekely, C.A. Knoblock, S. Gupta, M. Taheriyan, B. Wu, Exploiting semantics of web services for geospatial data fusion, in: Proceedings of the SIGSPATIAL International Workshop on Spatial Semantics and Ontologies, SSO 2011, Chicago, IL, 2011.
[23] M. Taheriyan, C.A. Knoblock, P. Szekely, J.L. Ambite, Semi-Automatically modeling web APIs to create linked APIs, in: Proceedings of the Linked APIs for the Semantic Web Workshop, LAPIS, 2012.
[24] M. Taheriyan, C.A. Knoblock, P. Szekely, J.L. Ambite, Rapidly integrating services into the linked data cloud, in: ISWC, Boston, MA, 2012, pp. 559–574.
[25] E.L. Lehmann, J.P. Romano, Testing Statistical Hypotheses, third ed., in: Springer Texts in Statistics, Springer, New York, 2005.
[26] P. Winter, Steiner problem in networks—a survey, Networks 17 (1987) 129–167.
[27] H. Takahashi, A. Matsuyama, An approximate solution for the Steiner problem in graphs, Math. Japonica 24 (1980) 573–577.
[28] L.T. Kou, G. Markowsky, L. Berman, A Fast Algorithm for Steiner trees, Acta Inform. 15 (1981) 141–145.
[29] K. Mehlhorn, A faster approximation algorithm for the Steiner problem in graphs, Inform. Process. Lett. 27 (3) (1988) 125–128.
[30] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan, Keyword searching and browsing in databases using BANKS, in: Proceedings of the 18th International Conference on Data Engineering, 2002, pp. 431–440.
[31] N. Craswell, Mean reciprocal rank, in: Encyclopedia of Database Systems, 2009, p. 1703.
[32] M. Arenas, P. Barcelo, L. Libkin, F. Murlak, Relational and XML Data Exchange, Morgan & Claypool, San Rafael, CA, 2010.
[33] Z. Bellahsene, A. Bonifati, E. Rahm, Schema Matching and Mapping, first ed., Springer, 2011.
[34] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, VLDB J. 10 (4) (2001) 334–350.
[35] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos, iMAP: Discovering complex semantic matches between database schemas, in: International Conference on Management of Data, SIGMOD, New York, NY, 2004, pp. 383–394.
[36] R. Fagin, L.M. Haas, M. Hernández, R.J. Miller, L. Popa, Y. Velegrakis, Clio: Schema mapping creation and data exchange, in: Conceptual Modeling: Foundations and Applications, 2009.

[37] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, Data Exchange: Semantics and query answering, Theoret. Comput. Sci. 336 (1) (2005) 89–124.

[38] B. Marnette, G. Mecca, P. Papotti, S. Raunich, D. Santoro, ++Spicy: an opensource tool for second-generation schema mapping and data exchange, in: Procs. VLDB, Seattle, WA, 2011, pp. 1438–1441.

[39] B. Alexe, B. ten Cate, P.G. Kolaitis, W.-C. Tan, Designing and refining schema mappings via data examples, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD'11, ACM, New York, NY, USA, 2011, pp. 133–144.

[40] Y. An, A. Borgida, R.J. Miller, J. Mylopoulos, A semantic approach to discovering schema mapping expressions, in: Proceedings of the 23rd International Conference on Data Engineering, ICDE, Istanbul, Turkey, 2007, pp. 206–215.

[41] C. Bizer, D2R MAP—a database to RDF mapping language, in: WWW (Posters), 2003.

[42] C. Bizer, R. Cyganiak, D2R server—publishing relational databases on the semantic web, in: Poster at the 5th International Semantic Web Conference, 2006.

[43] C. Bizer, A. Seaborne, D2RQ—treating non-RDF databases as virtual RDF graphs, in: ISWC2004 (posters), 2004.

[44] E. Muñoz, A. Hogan, A. Mileo, Triplifying wikipedia's tables, in: A.L. Gentile, Z. Zhang, C. d'Amato, H. Paulheim (Eds.), LD4IE@ISWC, in: CEUR Workshop Proceedings, vol. 1057, CEUR-WS.org, 2013.

[45] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, DBpedia: A nucleus for a web of open data, in: Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference, ISWC'07/ASWC'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 722–735.

[46] Z. Syed, T. Finin, Creating and exploiting a hybrid knowledge base for linked data, in: Agents and Artificial Intelligence, Springer, 2011, pp. 3–21.

[47] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor, Freebase: A collaboratively created graph database for structuring human knowledge, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD'08, ACM, New York, NY, USA, 2008, pp. 1247–1250.

[48] R. Parundekar, C.A. Knoblock, J.L. Ambite, Discovering concept coverings in ontologies of linked data sources, in: Proceedings of the 11th International Semantic Web Conference, ISWC, Boston, MA, 2012.

[49] M.J. Carman, C.A. Knoblock, Learning semantic definitions of online information sources, J. Artificial Intelligence Res. 30 (1) (2007) 1–50.

[50] Y. Kalfoglou, M. Schorlemmer, Ontology mapping: The state of the art, Knowl. Eng. Rev. 18 (1).

[51] S. Pavel, J. Euzenat, Ontology matching: State of the art and future challenges, IEEE Trans. Knowl. Data Eng. 25 (1) (2013) 158–176.

[52] M. Taheriyan, C. Knoblock, P. Szekely, J.L. Ambite, Y. Chen, Leveraging linked data to infer semantic relations within structured sources, in: Proceedings of the 6th International Workshop on Consuming Linked Data, COLD, 2015.